# PolyA: True Type Polymorphism for Mobile Ambients*

Torben Amtoft†
Kansas State University

Henning Makholm‡
Heriot-Watt University

J. B. Wells
Heriot-Watt University

2004-07-24

**Abstract**

Previous type systems for mobility calculi (the original Mobile Ambients, its variants and descendants, e.g., Boxed Ambients and Safe Ambients, and other related systems) offer little support for generic mobile agents. Previous systems either do not handle communication at all or globally assign fixed communication types to ambient names that do not change as an ambient moves around or interacts with other ambients. This makes it hard to type examples such as a messenger ambient that uses communication primitives to collect a message of non-predetermined type and deliver it to a non-predetermined destination.

In contrast, we present our new type system PolyA. Instead of assigning communication types to ambient names, PolyA assigns a type to each process P that gives upper bounds on (1) the possible ambient nesting shapes of any process P′ to which P can evolve, (2) the values that may be communicated at each location, and (3) the capabilities that can be used at each location. Because PolyA can type generic mobile agents, we believe PolyA is the first type system for a mobility calculus that provides type polymorphism comparable in power to polymorphic type systems for the λ-calculus. PolyA is easily extended to ambient calculus variants. A restriction of PolyA has principal typings.

*Partially supported by EC FP5/IST/FET grant IST-2001-33477 "DART", EPSRC grant GR/R41545/01, NSF grants 9806745 (EIA), 9988529 (CCR), and 0113193 (ITR), and Sun Microsystems equipment grant EDUD-7826-990410-US.

†Much of the work was done while Amtoft was at Heriot-Watt University paid by EC FP5/IST/FET grant IST-2001-33477 "DART".

‡Corresponding author. Email address: henning@makholm.net. Postal address: School of Mathematical and Computing Sciences, Heriot-Watt Univerity, Riccarton, Edinburgh EH14 4AS, Scotland, UK.

# 1 Introduction

Whereas the $\pi$-calculus [Milner 1999] is probably the most widely known calculus for communicating processes, the ambient calculus [Cardelli and Gordon 1998] has recently become important, because it adds reasoning about locations and mobility. In the ambient calculus, processes are located in *ambients*, locations which can be nested, forming a tree. Ambients can move, making the tree dynamic. Furthermore, only processes that are "close" to each other can exchange values.

## 1.1 The problem with ambient calculus type systems

Consider this process:

$$\mathtt{m}[\mathsf{in}\ \mathtt{s}.\mathsf{open}\ \mathtt{t}.(\mathtt{p},\mathtt{v}).\mathtt{p}.\langle\mathtt{v}\rangle.0] \mid \mathtt{s}[\mathtt{t}[\mathsf{in}\ \mathtt{m}.\langle\mathsf{in}\ \mathtt{r},\mathtt{d}\rangle.0] \mid \mathtt{r}[\mathsf{open}\ \mathtt{m}.(\mathtt{v}).\mathsf{out}\ \mathtt{v}.0]]$$

The example ambient named $\mathtt{m}$ is perhaps the simplest kind of *generic mobile agent*, namely a *messenger*. That is, $\mathtt{m}$ first goes somewhere looking for messages to deliver, then $\mathtt{m}$ collects a destination and a payload, and then $\mathtt{m}$ goes to that destination and delivers that payload. This rewriting sequence shows the behaviour of $\mathtt{m}$:

$$\begin{aligned}
& \mathtt{m}[\underline{\mathsf{in}\ \mathtt{s}}.\mathsf{open}\ \mathtt{t}.(\mathtt{p},\mathtt{v}).\mathtt{p}.\langle\mathtt{v}\rangle.0] \mid \mathtt{s}[\mathtt{t}[\mathsf{in}\ \mathtt{m}.\langle\mathsf{in}\ \mathtt{r},\mathtt{d}\rangle.0] \mid \mathtt{r}[\mathsf{open}\ \mathtt{m}.(\mathtt{v}).\mathsf{out}\ \mathtt{v}.0]] \\
\hookrightarrow\ & \mathtt{s}[\mathtt{m}[\mathsf{open}\ \mathtt{t}.(\mathtt{p},\mathtt{v}).\mathtt{p}.\langle\mathtt{v}\rangle.0] \mid \mathtt{t}[\underline{\mathsf{in}\ \mathtt{m}}.\langle\mathsf{in}\ \mathtt{r},\mathtt{d}\rangle.0] \mid \mathtt{r}[\mathsf{open}\ \mathtt{m}.(\mathtt{v}).\mathsf{out}\ \mathtt{v}.0]] \\
\hookrightarrow\ & \mathtt{s}[\mathtt{m}[\underline{\mathsf{open}\ \mathtt{t}}.(\mathtt{p},\mathtt{v}).\mathtt{p}.\langle\mathtt{v}\rangle.0 \mid \mathtt{t}[\langle\mathsf{in}\ \mathtt{r},\mathtt{d}\rangle.0]] \mid \mathtt{r}[\mathsf{open}\ \mathtt{m}.(\mathtt{v}).\mathsf{out}\ \mathtt{v}.0]] \\
\hookrightarrow\ & \mathtt{s}[\mathtt{m}[\underline{(\mathtt{p},\mathtt{v})}.\mathtt{p}.\langle\mathtt{v}\rangle.0 \mid \underline{\langle\mathsf{in}\ \mathtt{r},\mathtt{d}\rangle}.0] \mid \mathtt{r}[\mathsf{open}\ \mathtt{m}.(\mathtt{v}).\mathsf{out}\ \mathtt{v}.0]] \\
\hookrightarrow\ & \mathtt{s}[\mathtt{m}[\underline{\mathsf{in}\ \mathtt{r}}.\langle\mathtt{d}\rangle.0] \mid \mathtt{r}[\mathsf{open}\ \mathtt{m}.(\mathtt{v}).\mathsf{out}\ \mathtt{v}.0]] \\
\hookrightarrow\ & \mathtt{s}[\mathtt{r}[\underline{\mathsf{open}\ \mathtt{m}}.(\mathtt{v}).\mathsf{out}\ \mathtt{v}.0 \mid \mathtt{m}[\langle\mathtt{d}\rangle.0]]] \\
\hookrightarrow\ & \mathtt{s}[\mathtt{r}[\underline{(\mathtt{v})}.\mathsf{out}\ \mathtt{v}.0 \mid \underline{\langle\mathtt{d}\rangle}]] \\
\hookrightarrow\ & \mathtt{s}[\mathtt{r}[\mathsf{out}\ \mathtt{d}.0]]
\end{aligned}$$

In the first step, $\mathtt{m}$ moves into the sender $\mathtt{s}$; then $\mathtt{t}$ (which the sender has created as a temporary envelope for its message) moves into $\mathtt{m}$, where it gets dissolved. Then a local communication takes place between $\langle\mathsf{in}\ \mathtt{r},\mathtt{d}\rangle$ and $(\mathtt{p},\mathtt{v})$ which substitutes the destination path "$\mathsf{in}\ \mathtt{r}$" and the message $\mathtt{d}$ into the remainder of $\mathtt{m}$'s original code. In the fifth step $\mathtt{m}$ follows its assigned path and moves into the receiver $\mathtt{r}$; once arrived there it is dissolved, and its $\langle\mathtt{d}\rangle$ finally meets $(\mathtt{v})$ and the message $\mathtt{d}$ arrives at its destination.

Nearly all type systems for ambient calculi follow the example of the seminal system of Cardelli and Gordon [1999] and assign to each ambient name $\mathtt{a}$ a description of the communication that can happen within ambients named $\mathtt{a}$. Unfortunately, type systems based on this principle are inflexible about generic functionality. Consider the example process extended to have *two* possible execution paths, in that $\mathtt{m}$ can enter either of two senders:

$$\begin{aligned}
& \mathtt{m}[\mathsf{in}\ \mathtt{s}.\mathsf{open}\ \mathtt{t}.(\mathtt{p},\mathtt{v}).\mathtt{p}.\langle\mathtt{v}\rangle.0] \\
\mid\ & \mathtt{s}[\mathtt{t}[\mathsf{in}\ \mathtt{m}.\langle\mathsf{in}\ \mathtt{r},\mathtt{d}\rangle.0] \mid \mathtt{r}[\mathsf{open}\ \mathtt{m}.(\mathtt{v}).\mathsf{out}\ \mathtt{v}.0]] \quad (\mathtt{v}\ \textit{must be a name}) \\
\mid\ & \mathtt{s}[\mathtt{t}[\mathsf{in}\ \mathtt{m}.\langle\mathsf{in}\ \mathtt{q},\mathsf{out}\ \mathtt{d}\rangle.0] \mid \mathtt{q}[\mathsf{open}\ \mathtt{m}.(\mathtt{v}).\mathtt{v}.0]] \quad (\mathtt{v}\ \textit{must be a capability})
\end{aligned}$$

Here, the messenger $\mathtt{m}$ must be able to deliver two different types of payloads, *both* an ambient name *and* a capability. None of the previous type systems for ambient calculi allow this. In general, the previous type systems do not support the possibility that a mobile agent may carry non-predetermined types of data from location to location and deliver this data using communication primitives. Polymorphic type systems for the $\lambda$-calculus have no trouble with this kind of generic functionality.

In previous type systems for ambient calculi, generic mobile agents can be encoded by using extra ambient wrappers, one for each type of data to be delivered. Each location would be careful to only look inside arriving ambients with the correct name. However, this encoding is awkward and also loses the ability to predict whether the correct type of data is being delivered to each location, avoiding stuck states.

In solving this problem, a key observation is that the possible communication within $\mathtt{m}$ depends on which of the $\mathtt{s}$'s the ambient $\mathtt{m}$ is found inside.

## 1.2 Our solution – overview

To overcome the weaknesses of previous type systems for generic functionality, we present a new type system, PolyA. Types indicate the possible positions of capabilities, inputs, and outputs, and also represent upper bounds on the possible ambient nesting tree into which a process can evolve. Thus they look much like processes, as is also the case, e.g., for the types of Coppo and Dezani-Ciancaglini [2002].

Our type system's basic concept is the *shape predicate*. The actual definition is somewhat involved, partly due to the need of handling communication, so let us introduce the concept gently with a toy system where the only capability is "in":

$$\text{Toy shape predicates: } \sigma ::= 0 \mid (\sigma \mid \sigma) \mid a[\sigma] \mid \mathsf{in}\ a$$

A shape predicate's meaning is a set of terms, given by this matching relation:

$$\frac{\vdash P : \sigma}{\vdash a[P] : (\cdots \mid a[\sigma] \mid \cdots)} \qquad \frac{\vdash P : \sigma \qquad \vdash Q : \sigma}{\vdash P \mid Q : \sigma}$$

$$\frac{}{\vdash \mathsf{in}\ a.0 : (\cdots \mid \mathsf{in}\ a \mid \cdots)} \qquad \frac{}{\vdash 0 : \sigma}$$

(These rules are actually true in our full theory, but only the ones on the right are primitive).

With these rules we can derive the judgement $\vdash P_0 : \sigma_0$, where

$$\begin{aligned} P_0 &= a[\mathsf{in}\ b.0 \mid \mathsf{in}\ c.0] \mid b[d[\mathsf{in}\ a.0]] \mid c[e[\mathsf{in}\ a.0]] \\ \sigma_0 &= a[\mathsf{in}\ b \mid \mathsf{in}\ c] \mid b[d[\mathsf{in}\ a]] \mid c[e[\mathsf{in}\ a]] \end{aligned}$$

But we can also derive, say,

$$\vdash a[\mathsf{in}\ b.0] \mid a[\mathsf{in}\ c.0] : \sigma_0$$

— the matching rules do not care that the b and c on the top level are missing, nor that the a[in b | in c] part of the shape predicate is used twice.

PolyA *types* are shape predicates such that the set of terms matching a type is closed under reduction. The shape predicate $\sigma_0$ above is not a type, because

$$P_0 \hookrightarrow P_1 = \texttt{b[a[in c.0] | d[in a.0]] | c[}\cdots\texttt{]}$$

yet $\not\vdash P_1 : \sigma_0$. One type that $P_0$ does have is

$$\begin{aligned}\sigma_1 = \texttt{a[in b | in c] | b[a[in b | in c | d[in a]] | d[in a]]}\\ \texttt{| c[a[in b | in c | e[in a]] | e[in a]]}\end{aligned}$$

The a[···] predicate inside b still allows the in b. This must be so because shape predicates do not care about the number of identical items (unlike what is the case in [Teller et al. 2002]), so one of the terms matched by $\sigma_1$ is a[in b.0 | in b.0] | b[0], which reduces to b[a[in b]].

A more subtle point about $\sigma_1$ is that it *disallows* having an e inside an a inside a b, or a d inside an a inside a c. This example therefore illustrates the most basic kind of polymorphism possible: The same initial a ambient can evolve differently in different possible futures, and the type system can prove that those different futures do not interfere with each other.

This is the way polymorphism works in PolyA: An ambient can start out with a very small type such as a[in b | in c], and then when it chooses to move into b rather than c, it may change its type to a supertype, a[in b | in c | ···]. Seen from the viewpoint of the moving ambient, its type has evolved — though of course the new type has been present from the beginning somewhere in the overall type of the entire computation.

PolyA lets any supertype (i.e., a type that is matched by a larger set of terms) be used as a *polymorphic variant* if it appears in the right place of the overall typing. The overall typing contains all of the polymorphic variants that will ever be needed for each ambient in the particular context it is being typed in.

Some readers might think that this does not *look* like type polymorphism, because the various types for a are not substitution instances of a *parameterised* type. However, how one technically expresses the relation between the type for some generic code and the types for its concrete uses is not essential to the concept of genericity or polymorphism. What is important is that the type system supports reasoning about distinct uses of the same generic code. We achieve what Cardelli and Wegner [1985] called "the purest form of polymorphism: the same object or function can be used uniformly in different type context without changes, coercions or any kind of run-time tests or special encodings of representations". As an example of the generality of the concept of polymorphism, type polymorphism in the λ-calculus includes methods such as intersection types [Coppo and Dezani-Ciancaglini 1980], where all of the type instances are present from the beginning and no ∀-quantifiers are needed.

Other features of PolyA are as follows. There are singleton types of ambient names and explicit dependencies on communication, as illustrated by this judgement:

$$\vdash (\langle \texttt{a} \rangle.\texttt{0} \mid (\texttt{x}).\texttt{x[0]}) : (\langle \texttt{a} \rangle.\texttt{0} \mid (\texttt{x}).\texttt{x[0] | a[0]})$$

Types can be infinitely deep trees, e.g.:

$$\vdash \texttt{!a[!in a.0]} : \text{letrec } \texttt{X} = (\texttt{a[X]} \mid \text{in a.0}) \text{ in } \texttt{a[X]}$$

We only consider types that can be given a finite term representation.

Unlike previous type systems, there are no type assumptions for ambient names. Instead, information on the topics of conversation inside various ambients is put in the types of processes. Also unlike previous type systems, there are no type assumptions for the types of the bound variables of input processes.

PolyA can assign the following type to the example containing the generic messenger from page 2:

```
letrec X9 = (p, v).(p | ⟨{v}⟩) | in s | open t
in m[X9]
 | s[letrec X4 = (X7) | (X9) | m[X5] | t[X7] | (v).(out v)
                | amb r | in r | open m | out d | ⟨{d}⟩
            X5 = (X7) | (X9) | r[X4] | t[X7] | amb m | in r | ⟨{d}⟩
            X7 = in m | ⟨<in r>, {d}⟩
     in m[X5] | r[X4] | t[X7] end]
 | s[letrec X1 = (X8) | (X9) | m[X2] | t[X8] | (v).(v)
                | amb q | in q | open m | out d | ⟨<out d>⟩
            X2 = (X8) | (X9) | q[X1] | t[X8] | amb m | in q | ⟨<out d>⟩
            X8 = in m | ⟨<in r>, <out d>⟩
     in m[X2] | r[X1] | t[X8] end]
end
```

This type is also shown graphically in Figure 1.

This proves that the example process has only well defined behaviour, something which no previous type system for ambients can do.

This type is not *principal*, that is, it is possible to construct more precise PolyA types for the messenger examples. It has been artificially restricted to not give any information about the ordering of, say, the capabilities `in s` and `open t` within `m`. There is also a type that includes the fact that `open t` will only be available after `in s` has been executed, and similarly for the other capability sequences in the term. Such types tend to be more complex and harder to read than less precise types.

As another example, `a[in b.in c.0] | b[c[0]] | c[open a.0]` has a PolyA type proving that `a` will never be opened. This ability to track the sequencing of actions was pioneered by Amtoft et al. [2001, 2002].

## 1.3 Other related work

Although not type-based, several papers have explored letting the analysis of an ambient subprocess depend on its possible contexts — a task which requires an estimate of the possible shapes of the ambient tree structure. None of these handle communication, however, so none can prove the safety of our example

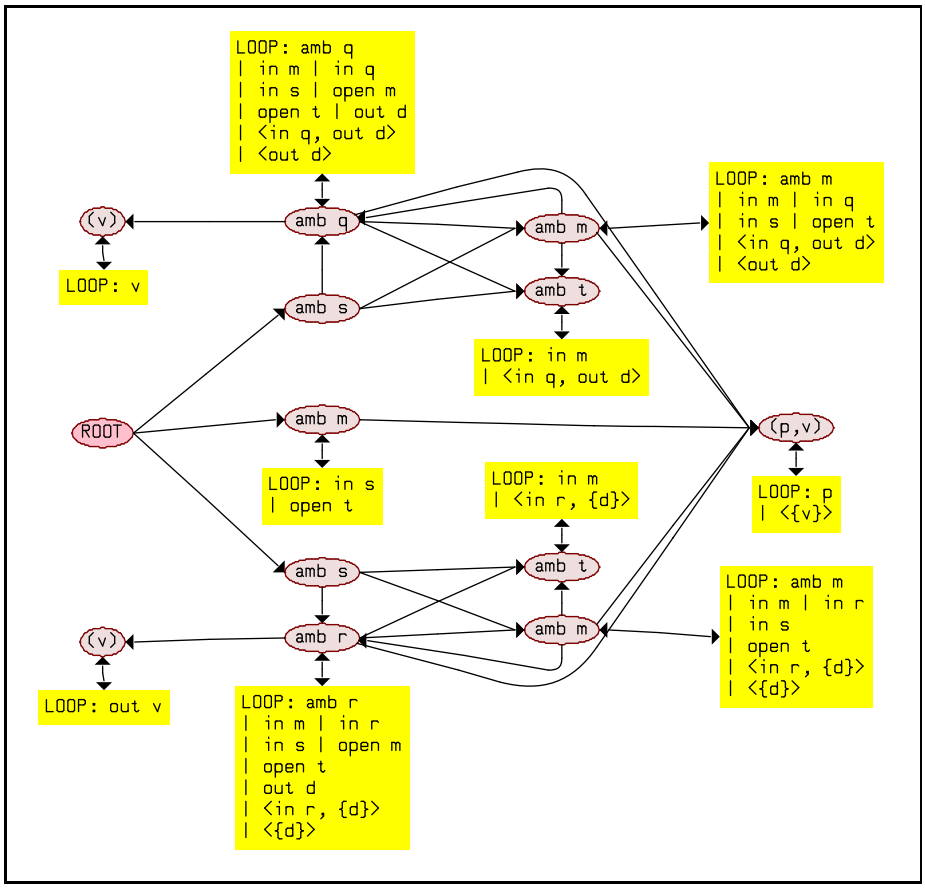


*Figure 1: A graphical representation of the type for the polymorphic messenger example. For readability, this figure deviates slightly from the shape graph formalism of Section 3.1. Bundles of length-one loops of the form* $X \xrightarrow{\pi} X$ *are represented as rectangles labelled "LOOP". The labels of non-loop edges are shown inside the target nodes (which would not be possible if two non-loop edges with the same target had different labels, but that happens not to be the case here).*
*The graph has been produced from a machine-generated source, plus some manual layout hints, using the* VCG *graph layout tool [Sander 1994].*

polymorphic messenger. With shape grammars [Nielson and Nielson 2000], a set of grammars is returned such that at any step, the current process can be described by one of these grammars. The analysis is very precise, but potentially also very expensive. In Kleene analysis [Nielson et al. 2000], a 3-valued logic is used to estimate the possible shapes. The framework allows for trade-offs w.r.t. precision versus costs. The abstract interpretation system of Levi and Maffeis [2001] keeps track of the context "one level up". This is sufficient to achieve a quite precise analysis, yet is "only" polynomial ($n^7$).

Polymorphic type systems for process calculi are not new; for example Turner [1995] defined a polymorphic type system for the $\pi$-calculus (which was later generalised using intersection types by Maffeis [2004]). These systems are different from, and orthogonal to, the form of polymorphism provided by PolyA, in that it is *communication* rather than *movement* that that causes polymorphic variants to be created. PolyA does not support the polymorphism-by-communication paradigm (and neither does any type system for a mobility calculus with locations that we are aware of); we leave it to future work to try to combine the strengths of these two principles.

### 1.4 Summary of contributions (conclusion)

- We present PolyA, the first type system for the ambient calculus that is flexible enough to type generic mobile agents.
- We explain how PolyA types can be used not just to check basic type safety but also to give precise answers to various questions about process behaviour of interest for other reasons, e.g., security.
- We prove subject reduction (Theorem 5.7(1)) and the decidability of type checking (Proposition 3.5) for PolyA.
- We prove principal typings (Theorem 5.13) for a useful restriction of PolyA.
- We illustrate how to extend PolyA to support the cross-ambient communication of Boxed Ambients [Bugliesi et al. 2001], the co-capabilities of Safe Ambients [Levi and Sangiorgi 2000], and the process (not ambient) mobility capability of $M^3$ [Coppo et al. 2003].

We do not consider type reconstruction here, but in other work [Makholm and Wells 2004] we have developed a practical type inference algorithm for a useful restriction of PolyA.

### 1.5 Acknowledgements

The design of PolyA benefited from helpful discussions with Mario Coppo, Mariangiola Dezani, and Elio Giovannetti.

A preliminary version of this paper appeared as Amtoft et al. [2004].

## 2 The ambient calculus

To present the underlying ideas clearly, we work in a calculus without name restriction. Name restriction will be added in Section 6. Figure 2 defines the syntax and semantics of our base calculus. Whenever it has been defined that some (meta)variable letter, say "$x$", ranges over a given set of objects, the notation $\boxed{x}$ shall mean that set of objects.

**Syntax:**

$$
\begin{array}{rl}
\text{Names:} & a, b ::= a \mid b \mid c \mid \cdots \\
\text{Opcodes:} & O ::= \mathsf{in} \mid \mathsf{out} \mid \mathsf{open} \mid \mathsf{amb} \\
\text{Capabilities:} & C ::= a \mid O\,a \mid \bullet \\
\text{Messages:} & M, N ::= C \mid M.N \mid \varepsilon \\
\text{Prefixes:} & p ::= M \mid \langle \vec{M} \rangle \mid (\vec{a}) \\
\text{Processes:} & P, Q, R ::= p.P \mid\ !P \mid (P \mid Q) \mid 0
\end{array}
$$

See main text for further syntactic restrictions (scoping).

**Process equivalence:**

$$\overline{P \mid Q \equiv Q \mid P} \qquad \overline{P \mid (Q \mid R) \equiv (P \mid Q) \mid R} \qquad \overline{0 \mid P \equiv P}$$

$$\overline{!P \equiv P \mid\ !P} \qquad \overline{!0 \equiv 0} \qquad \overline{(M.N).P \equiv M.(N.P)} \qquad \overline{\varepsilon.P \equiv P} \qquad \overline{P \equiv P}$$

$$\frac{P \equiv Q}{p.P \equiv p.Q} \qquad \frac{P \equiv Q}{P \mid R \equiv Q \mid R} \qquad \frac{P \equiv Q}{!P \equiv\ !Q} \qquad \frac{Q \equiv P}{P \equiv Q} \qquad \frac{P \equiv Q \qquad Q \equiv R}{P \equiv R}$$

**Substitution:**
A **term substitution** $\mathcal{S}$ is a (total) function from names to messages such that $\mathcal{S}(a) \neq a$ for only finitely many $a$'s. We often notate it $\mathcal{S} = [a_1 \mapsto M_1, \ldots, a_k \mapsto M_k]$, understanding implicitly that $\mathcal{S}(a) = a$ when $a$ is not one of the $a_i$'s. Shorter notations are $[a_i \mapsto M_i]_{1 \le i \le k}$ or $[a \mapsto M_a]_{a \in A}$.

For messages:
$$\mathcal{S}\,(M.N) = (\mathcal{S}\,M).(\mathcal{S}\,N) \qquad \mathcal{S}\,\varepsilon = \varepsilon$$
$$\mathcal{S}\,a = \mathcal{S}(a) \qquad \mathcal{S}\,\bullet = \bullet$$
$$\mathcal{S}\,(O\,a) = \begin{cases} O\,\mathcal{S}(a) & \text{if } \mathcal{S}(a) \text{ is a name} \\ \bullet & \text{otherwise} \end{cases}$$

For other prefixes: $\mathcal{S}\,\langle M_1, \ldots, M_k \rangle = \langle \mathcal{S}\,M_1, \ldots, \mathcal{S}\,M_k \rangle$
$$\mathcal{S}\,(a_1, \ldots, a_k) = \begin{cases} (a_1, \ldots, a_k) & \text{if } \mathcal{S}(a_i) = a_i \text{ for all } i \\ \bullet & \text{otherwise} \end{cases}$$

For terms:
$$\mathcal{S}\,(p.P) = (\mathcal{S}\,p).(\mathcal{S}\,P) \qquad \mathcal{S}\,(!P) = !(\mathcal{S}\,P)$$
$$\mathcal{S}\,(P \mid Q) = (\mathcal{S}\,P) \mid (\mathcal{S}\,Q) \qquad \mathcal{S}\,0 = 0$$

**Reduction rules:**

$$\overline{a\,[\mathsf{in}\ b.P \mid Q] \mid b\,[R] \hookrightarrow b\,[a\,[P \mid Q] \mid R]}$$

$$\overline{b\,[a\,[\mathsf{out}\ b.P \mid Q] \mid R] \hookrightarrow a\,[P \mid Q] \mid b\,[R]} \qquad \overline{a\,[P] \mid \mathsf{open}\ a.Q \hookrightarrow P \mid Q}$$

$$\overline{\langle M_1, \ldots, M_n \rangle.P \mid (a_1, \ldots, a_n).Q \hookrightarrow P \mid [a_i \mapsto M_i]_{1 \le i \le n} Q}$$

$$\frac{P \hookrightarrow Q}{a\,[P] \hookrightarrow a\,[Q]} \qquad \frac{P \hookrightarrow Q}{P \mid R \hookrightarrow Q \mid R} \qquad \frac{P \equiv P' \qquad P' \hookrightarrow Q' \qquad Q' \equiv Q}{P \hookrightarrow Q}$$

*Figure 2: Syntax and semantics of the ambient calculus*

The syntactic category of *prefixes* is not in traditional ambient calculus formulations. Our calculus treats ambient boundaries as capabilities; "amb a" is the capability that creates an ambient named a when executed. In our formulation, an ambient with contents P is written "amb a.P". The traditional notation "a[P]" is syntactic sugar for amb a.P; we use this whenever convenient. The capability amb a can in principle be passed in a message. We allow this more because it is syntactically convenient than because we expect processes to actually do it. Indeed, the main results for our type system (Theorem 5.7(2); Proposition 5.18) discriminate against programs that communicate ambient-creation capabilities; this allows more precise typings of programs that do not communicate amb capabilities.[1]

Like Bugliesi et al. [2001], we have synchronous output $\langle \vec{M} \rangle$.P; asynchronous output (as in the original ambient calculus) is a special case with $P = 0$.

The parentheses around parallel composition P | Q can be omitted when unambiguous. Prefixes bind tighter than parallel composition.

The special capability "•" is not supposed to be found in the initial term. It signifies a substitution result that would otherwise be syntactically invalid. For example, the term ⟨in c⟩ | (b).in a.open b.0 reduces to in a.•.0 instead of the (hypothetical) "in a.open (in c).0". Traditional ambient calculus accounts usually leave such a communication result undefined, implicitly understanding that the system would crash either at the communication time or when the ill-formed capability executes after the in a capability has fired. Our approach supports both views, according to how one interprets •. In either case, it is technically convenient to reify the failure as a special term.

The symbol • does not have any reduction rules associated with it. As far as our theory is concerned it just sits there. Likewise, there are no reduction rules for placeholder capabilities of the form "a". In applications, one may or may not want to treat it as an error if such a capability shows up in a place where it wants to be executed. A PolyA type conservatively approximates *whether* one of these capabilities may occur, but the type system user must decide what to do if this happens.

**Convention 2.1.** A term P is **well formed** iff its free names are distinct from the names bound by any "$(\vec{a})$" within the term and it does not contain any nested bindings of the same name. We consider only well formed terms.

Convention 2.1 does not limit expressiveness. Any program (term) in a more conventional ambient calculus formulation that allows $\alpha$-conversion has a well formed $\alpha$-variant which can be used in our type system.[2]

[1] One may get a type system that does not discriminate by removing the special cases for amb from Definitions 4.9 and 4.10. But then, e.g., a[in b] | b[open a | ⟨ε⟩] | (c).c[0] would not have a •-free type anymore.

[2] If one later discovers that one really wanted to use another variant, for example because the free names are only learned incrementally during compositional analysis, it is a simple matter to correct it retroactively by swapping all instances of one name with a fresh new one, in terms and types alike.

The convention ensures that our reduction rules will never perform a substitution where there is a risk of name capture by $(\vec{a})$ bindings. Reductions preserve well-formedness, because it is syntactically impossible for a substitution to inject a $(\vec{a})$ within the body of another $(\vec{a})$. (This is in contrast to the $\lambda$-calculus, where substitutions routinely insert $\lambda$-abstractions into other abstractions). Because of this, we do not need to recognise $\alpha$-equivalence for $(\vec{a}).P$. This is a significant technical simplification, because for many purposes we can treat $(\vec{a})$ as any other action, without needing special machinery for $\alpha$-equivalence of the bound names. Once we introduce name restriction in Section 6, we *will* need to handle $\alpha$-conversion of $\nu$-bound names; Convention 2.1 explicitly applies only to $(\vec{a})$ binders.

Figure 2 contains no provisions for avoiding name capture in $\mathcal{S}(\vec{a})$ — this is handled by Convention 2.1. The $\bullet$ possibility for $\mathcal{S}(\vec{a})$ is never supposed to be used; substitutions leading to it will not arise by our rules.

**Lemma 2.2.** *Substitution is compatible with term equivalence:* $P \equiv Q$ *implies* $\mathcal{S}P \equiv \mathcal{S}Q$.

# 3 Shape predicates

The following pseudo-grammar defines the (abstract) syntax of our type system:

$$
\begin{array}{lll}
\text{Message types:} & \mu ::= \{C_1, C_2, \cdots, C_k\}* & (C_i\text{'s all different}, k \geq 1) \\
& \quad | \quad <C_1.C_2.\cdots.C_k> & (C_i\text{'s all different}, k \geq 0) \\
& \quad | \quad \{a\} & \\
\text{Prefix types:} & \pi ::= C \;\big|\; (\vec{a}) \;\big|\; \langle\vec{\mu}\rangle & \\
\text{Shape predicates:} & \sigma ::= (\pi_1.\sigma_1 \mid \cdots \mid \pi_k.\sigma_k) & (k \geq 1) \\
& \quad | \quad 0 &
\end{array}
$$

**Definition 3.1 (matching of shape predicates).** *The following rules define the relations* $\vdash M : \mu$, $\vdash p : \pi$, *and* $\vdash P : \sigma$*:*

$$\frac{M \notin \boxed{a} \qquad M.0 \equiv C'_1.\cdots.C'_n.0 \qquad \{C'_1, ..., C'_n\} \subseteq \{C_1, ..., C_k\}}{\vdash M : \{C_1, \ldots, C_k\}*}\ \text{KleeneStar}$$

$$\frac{M \notin \boxed{a} \qquad M.0 \equiv C_1.\cdots.C_k.0}{\vdash M : <C_1.\cdots.C_k>}\ \text{Sequenced} \qquad \frac{}{\vdash a : \{a\}}\ \text{Name}$$

$$\frac{}{\vdash C : C}\ \text{Cap} \qquad \frac{}{\vdash (\vec{a}) : (\vec{a})}\ \text{Recv} \qquad \frac{\vdash M_1 : \mu_1 \qquad \cdots \qquad \vdash M_k : \mu_k}{\vdash \langle M_1, \ldots, M_k\rangle : \langle \mu_1, \ldots, \mu_k\rangle}\ \text{Send}$$

$$\frac{\vdash p : \pi \qquad \vdash P : \sigma}{\vdash p.P : (\cdots \mid \pi.\sigma \mid \cdots)}\ \text{Pfx} \qquad \frac{\vdash M.(N.P) : \sigma}{\vdash (M.N).P : \sigma}\ \text{Seq} \qquad \frac{\vdash P : \sigma}{\vdash \varepsilon.P : \sigma}\ \text{Nop}$$

$$\frac{\vdash P : \sigma \qquad \vdash Q : \sigma}{\vdash P \mid Q : \sigma}\ \text{Par} \qquad \frac{}{\vdash 0 : \sigma}\ \text{Null} \qquad \frac{\vdash P : \sigma}{\vdash !P : \sigma}\ \text{Bang}$$

The side conditions $M \notin \boxed{a}$ and $M.0 \equiv C_1.\cdots.C_k.0$ on rules KleeneStar and Sequenced amount to specifying that these two forms of message types are matched modulo associativity of "." and neutrality of "$\varepsilon$" — with the exception that messages that are raw names (i.e., "$a$" as opposed to "$a.\varepsilon$" or "in $a$") are handled specially. They are matched only by the message type $\{a\}$. The property that $a$ and $a.\varepsilon$ cannot have *any* common type is essential to our approach. It allows us to decide whether $[a \mapsto M](\mathsf{in}\ a)$ will produce $\bullet$ or an actual capability, without knowing anything else about $M$ than its type.

Our language of message types is rather restricted, but it has been carefully constructed to allow the proofs to go through — in particular, for principal types to exist, we need to ensure that each message type has only finitely many subtypes (Lemma C.3). This would not be the case if message types had been, say, regular expressions over capabilities.

No prefix type matches a prefix of the form $M$ where $M$ is not a naked capability. Such prefixes are handled by rules Seq and Nop.

**Theorem 3.2.** *If* $P \equiv Q$ *then* $\vdash P : \sigma \Leftrightarrow \vdash Q : \sigma$ *for all* $\sigma$.

Proof. By induction on the derivation of $P \equiv Q$. □

**Definition 3.3.** *The* ***meaning*** *of a shape predicate (message type, prefix type) is the set of terms (messages, prefixes) that match it:*

$$[\![\mu]\!] = \{ M \mid \vdash M : \mu \} \qquad [\![\pi]\!] = \{ p \mid \vdash p : \pi \} \qquad [\![\sigma]\!] = \{ P \mid \vdash P : \sigma \}$$

**Definition 3.4.** *Define the following* ***containment*** *relations:*

$$\mu \leq \mu' \iff [\![\mu]\!] \subseteq [\![\mu']\!] \qquad \pi \leq \pi' \iff [\![\pi]\!] \subseteq [\![\pi']\!] \qquad \sigma \leq \sigma' \iff [\![\sigma]\!] \subseteq [\![\sigma']\!]$$

Each of the three containment relations is a preorder (transitive and reflexive). Containment of shape predicates is not antisymmetric, however. For example, amb a.amb b.0 and amb a.amb b.0 | amb a.0 have the same meaning, but it would be technically inconvenient (and not give any real benefit) to insist on equating shape predicates with equal meanings.

## 3.1 Recursive shape predicates

Our strategy in analysing a term is to look for a shape predicate describing all of its possible computational futures. Because many terms can create arbitrarily deep nestings of ambients (e.g., !a[!in a.0]), the finite trees we have used for shape predicates so far are not up to the task[3]. We need infinite shape predicates. We should, however, restrict ourselves to infinite shape predicates with finite *representations* — in other words, regular trees.

[3]This happens even for terminating terms such as b[in a.0] | a[open b.0], because shape predicates cannot distinguish them from !b[!in a.0] | !a[open b.0]. Thus, nearly every nontrivial use of open will need recursive $\sigma$'s. As already observed by Cardelli et al. [1999], open often complicates analysis significantly.

There are several regular tree representations that we could have used. We believe it is technically most convenient (and intuitive) to view regular trees as *graphs*. Therefore, we retroactively replace the abstract syntax for shape predicates with:

$$\begin{array}{ll} \text{Node identifiers: } X, Y, Z ::= & \mathtt{X1} \mid \mathtt{X2} \mid \mathtt{X3} \mid \cdots \\ \text{Edges:} & e ::= X \xrightarrow{\pi} Y \\ \text{Shape graphs:} & G \in \mathcal{P}_{\text{fin}}(\boxed{e}) \\ \text{Shape predicates:} & \sigma ::= \langle X \mid G \rangle \end{array}$$

A shape predicate is now a shape graph together with a pointer to a distinguished *root* node. The version of the Pfx rule that works with this notation is

$$\frac{\vdash p : \pi \qquad X \xrightarrow{\pi} Y \in G \qquad \vdash P : \langle Y \mid G \rangle}{\vdash p.P : \langle X \mid G \rangle}\ \mathsf{Pfx}$$

Theorem 3.2 is still true with this formulation, because it was proven by induction on term equivalence rather than shape-predicate structure.

In general, defining some property for shape graphs implicitly defines it for shape predicates: The shape predicate $\langle X \mid G \rangle$ has the property iff $G$ has.

This graph-based formulation is the basis for our formal development. However, even though graphs are an intuitive way of *thinking* about regularly infinite shape predicates, they are less convenient for *writing down* shape predicates, at least in a human-friendly form. Figure 3 defines a more tree-like textual notation for shape graphs for use in examples.

**Proposition 3.5.** *The relations of Definition 3.1 are effectively (and efficiently) decidable when shape predicates are given as graphs.*

Proof. It is easy to see how to decide $\vdash M : \mu$ and $\vdash p : \pi$, so we argue only for the case of $\vdash P : \sigma$.

Without loss of generality, we can assume that $P$ does not contain any subterm that matches the conclusion of either Seq or Nop – if it does, it is always easy to use local $\equiv$-rewritings to make them go away; then Theorem 3.2 justifies matching the rewritten term instead of the original one.

Now proceed by induction on the structure of $P$. The cases corresponding to parallel composition, replication and $\mathtt{0}$ are easy: Here the syntax dictates that a derivation of $\vdash P : \sigma$, if it exists, must end with the Par, Null, or Bang rule, respectively.

All other forms of $P$ must be matched by the Pfx rule. Here the problem is that there may be more than one applicable instance of the rule. However, the since the shape graph $G$ is supposed to be finite, there is only finitely many recursive instances to check. □

The algorithm naively implied by this proof may have a running time that is exponential in the depth of $P$. However, it is easily seen that only $|P| \cdot |G|$ different judgements may ever be considered during the execution; if one memoizes the answer for judgements that have already been attempted, the complexity of matching becomes practically tractable.

This is the syntax of **shape expressions**:

Shape expressions: $V ::= U \mid X \mid \mathsf{letrec}\ X_1 = U_1; \ldots; X_n = U_n\ \mathsf{in}\ X_i$
Shape summands: $U ::= 0 \mid (U \mid U) \mid \pi \mid (\pi_1 \mid \cdots \mid \pi_n).V \mid (X)$

As additional syntactic sugar, $\mathsf{letrec}\ \cdots\ \mathsf{in}\ U$ stands for $\mathsf{letrec}\ \cdots; X = U\ \mathsf{in}\ X$ where $X$ is fresh. $\pi.V$ stands for $(\pi).V$, and $a[V]$ stands for $(\mathsf{amb}\ a).V$.

To convert a shape expression to a graph-shaped shape predicate, first replace each $U$ of the form $(X)$ with the right-hand side of the innermost in-scope letrec binding for $X$. It is an error if no such binding exist, or if the unfolding does not terminate. ($V$'s of the form $X$ are not touched at this stage). Then $\alpha$-rename the entire shape expression such that no $X$ is bound by two different letrec's, and apply the function $(\cdot)^*$ defined by:

a) $V^*$ is a shape predicate:

$$X^* = \langle X \mid \varnothing \rangle \qquad U^* = \langle X \mid U^*_X \rangle \text{ where } X \text{ is fresh}$$
$$(\mathsf{letrec}\ X_1 = U_1; \ldots; X_n = U_n\ \mathsf{in}\ X_i)^* = \langle X_i \mid U_1{}^*_{X_1} \cup \cdots \cup U_n{}^*_{X_n} \rangle$$

b) $U^*_X$ is a shape graph:

$$0^*_X = \varnothing \qquad (U_1 \mid U_2)^*_X = (U_1)^*_X \cup (U_2)^*_X$$
$$\pi^*_X = \{X \xrightarrow{\pi} X\} \qquad ((\pi_1 \mid \cdots \mid \pi_n).V)^*_X = \{X \xrightarrow{\pi_i} X' \mid 1 \le i \le n\} \cup G \text{ where } \langle X' \mid G \rangle = V^*$$

Note that the parentheses in $U ::= (X)$ are important; they distinguish between "$\pi.X$", which says to insert an edge going to node $X$ itself, and "$\pi.(X)$", which says to insert an edge to a fresh node that happens to behave like $X$. This can make a difference for whether the shape graph satisfies certain criteria that we'll define later.

*Figure 3: Shape expressions: a tree-like notation for recursive shape predicates*

**Definition 3.6.** *Two shape graphs $G_1$ and $G_2$ are **equivalent**, written $G_1 \approx G_2$, iff $[\![\langle X \mid G_1 \rangle]\!] = [\![\langle X \mid G_2 \rangle]\!]$ for all $X$.*

## 3.2 Effective characterisation of containment

Definition 3.4 defined a containment order on shape predicates (and message and prefix types) in an intuitively appealing way, but it did not provide a decision procedure. This subsection develops a more effective characterisation.

**Definition 3.7.** *Let $R$ be a relation between shape predicates. $R$ is a **shape simulation** iff $\langle X \mid G \rangle\ R\ \langle X' \mid G' \rangle$ and $X \xrightarrow{\pi} Y \in G$ imply that there is $\pi' \ge \pi$ and $Y'$ such that $X' \xrightarrow{\pi'} Y' \in G'$ and $\langle Y \mid G \rangle\ R\ \langle Y' \mid G' \rangle$.*

**Lemma 3.8.** *Let $P$ be an arbitrary term. If $R$ is a shape simulation and $\sigma\ R\ \sigma'$, then $\vdash P : \sigma$ implies $\vdash P : \sigma'$.*

Proof. By induction on the structure of $P$. □

**Lemma 3.9.** *For any message type $\mu$ there exists a message $\lceil\mu\rceil$ such that $\vdash \lceil\mu\rceil : \mu' \iff \mu \leq \mu'$.*

Proof. Let $\lceil\{C_1, \ldots, C_k\}*\rceil = C_1.C_1.C_2.C_2.\cdots.C_k.C_k$ (with two copies of each capability). Let $\lceil<C_1.\cdots.C_k>\rceil = C_1.C_2.\cdots.C_k.\varepsilon$. Let $\lceil\{a\}\rceil = a$. □

**Corollary 3.10.** *For any prefix type $\pi$ there exits a prefix $\lceil\pi\rceil$ such that $\vdash \lceil\pi\rceil : \pi' \iff \pi \leq \pi'$.*

**Proposition 3.11.** *The shape containment relation $\leq$ is a shape simulation.*

Proof. Assume $\langle X \mid G\rangle \leq \langle X' \mid G'\rangle$ and $X \xrightarrow{\pi} Y \in G$. Let $\mathbf{Y} = \{\, Y' \mid X' \xrightarrow{\pi'} Y' \in G', \pi \leq \pi' \,\}$.

Set $P_0 = 0$, and iterate the following operation for $i = 0, 1, 2, \ldots$: Let $\mathbf{Y}_i = \{\, Y' \in \mathbf{Y} \mid \vdash P_i : \langle Y' \mid G'\rangle \,\}$. Choose $Y'_i \in \mathbf{Y}_i$ and $Q_i \in [\![\langle Y \mid G\rangle]\!] \setminus [\![\langle Y'_i \mid G'\rangle]\!]$. If that is not possible, then we are done; continue below. Otherwise let $P_{i+1} = P_i \mid Q_i$ and repeat for $i + 1$. Because $\mathbf{Y}_{i+1}$ is strictly smaller than $\mathbf{Y}_i$ (at least $Y'_i$ is missing) and $\mathbf{Y}_0 = \mathbf{Y}$ is finite (because $G'$ is), the process will stop eventually.

When the iteration stops after step $n$ we have $P_n = 0 \mid Q_0 \mid \cdots \mid Q_{n-1}$ with each $Q_i \in [\![\langle Y \mid G\rangle]\!]$. Therefore $P_n \in [\![\langle Y \mid G\rangle]\!]$. Set $p = \lceil\pi\rceil$ (from Corollary 3.10). Then $p.P_n \in [\![\langle X \mid G\rangle]\!] \subseteq [\![\langle X' \mid G'\rangle]\!]$, so $G'$ must contain $X' \xrightarrow{\pi'} Y'$ for some $Y'$ such that $P_n \in [\![\langle Y' \mid G'\rangle]\!]$. Furthermore $p = \lceil\pi\rceil \in [\![\pi']\!]$ so $\pi \leq \pi'$, $Y' \in \mathbf{Y}$, and $Y' \in \mathbf{Y}_n$.

But then it must be the case that $\langle Y \mid G\rangle \leq \langle Y' \mid G'\rangle$ – because otherwise we could have chosen $Y'$ as $Y'_n$ and continued the iteration. Therefore $X' \xrightarrow{\pi'} Y'$ satisfies the requirements on the definition of a shape simulation. □

**Theorem 3.12.** *Shape containment $\leq$ is the largest shape simulation; it is the union of all shape simulations.*

Proof. Lemma 3.8 implies that every shape simulation is contained in $\leq$; the theorem now follows from Proposition 3.11. □

Theorem 3.12 tells us that to prove that $\sigma \leq \sigma'$ it is sufficient to find a shape simulation $R$ such that $\sigma\ R\ \sigma'$. This principle can be used to decide $\leq$:

**Lemma 3.13.** *The relation $\mu \leq \mu'$ is effectively decidable.*

Proof. There are 9 cases, according to the syntactic shapes of $\mu$ and $\mu'$. Each of them is easy. □

**Lemma 3.14.** *The relation $\pi \leq \pi'$ is effectively decidable.*

Proof. $\pi \leq \pi'$ if and only if at least one of the following conditions hold:

1. $\pi = \pi'$, or
2. $\pi = \langle \mu_1, \ldots, \mu_k\rangle$ and $\pi' = \langle \mu'_1, \ldots, \mu'_k\rangle$ and $\forall i : \mu_i \leq \mu'_i$. □

**Lemma 3.15.** *For any shape graph* $G$*, define its* ***support****:*

$$\overline{G} = \{\, \langle X \mid G \rangle \mid X \xrightarrow{\pi} Y \in G \,\} \cup \{\, \langle Y \mid G \rangle \mid X \xrightarrow{\pi} Y \in G \,\}$$

*Let* $G$ *and* $G'$ *be arbitrary shape graphs and let* $R$ *be any shape simulation. Then* $R \cap (\overline{G} \times \overline{G'})$ *is also a shape simulation.*

**Proposition 3.16.** *The relation* $\langle X \mid G \rangle \leq \langle X' \mid G' \rangle$ *can be decided effectively (actually, in polynomial time).*

Proof. Because of Lemma 3.14 it is easy to decide whether a (finite) relation between shape predicates is a shape simulation.

If $\langle X \mid G \rangle \notin \overline{G}$ or $\langle X' \mid G' \rangle \notin \overline{G'}$, then $\langle X \mid G \rangle \leq \langle X' \mid G' \rangle$ is easy to decide.

Otherwise start with the relation $\overline{G} \times \overline{G'}$ and try to make it into a shape simulation by excluding pairs from it whenever their presence would lead to Definition 3.7 failing. Removing even more pairs can never cause any of the pairs we have previously removed to become admissible again, so eventually the algorithm will stop, having computed the largest shape simulation that is a subset of $\overline{G} \times \overline{G'}$. Lemma 3.15 and Theorem 3.12 tell us that this must actually be $(\leq) \cap (\overline{G} \times \overline{G'})$, so in it we can look up directly whether $\langle X \mid G \rangle \leq \langle X' \mid G' \rangle$. □

It is worth noticing that shape simulations treat $(\vec{a})$ just like any other prefix type. Thus $\leq$ treats the "result" type covariantly (like Zimmer [2000]), whereas the input position in PolyA is a list of names and thus essentially invariant.

# 4 Substitution and shape predicates

**Definition 4.1.** *A* ***type substitution*** $\mathcal{T}$ *is a function from names to message types such that* $\mathcal{T}(a) \neq \{a\}$ *for only finitely many* $a$*'s. Like term substitutions, type substitutions may be written as* $[a_1 \mapsto \mu_1, \ldots, a_k \mapsto \mu_k]$ *or* $[a \mapsto \mu_a]_{a \in A}$.

**Definition 4.2.** *Extend matching of message types pointwise to type substitutions:* $\vdash \mathcal{S} : \mathcal{T}$ *iff* $\vdash \mathcal{S}(a) : \mathcal{T}(a)$ *for all* $a$.

**Proposition 4.3.** *There is a natural operation of type substitutions on capabilities such that* $\mathcal{T}C$ *is a message type with the property*

$$[\![\mathcal{T}C]\!] = \{\, \mathcal{S}C \mid \vdash \mathcal{S} : \mathcal{T} \,\}$$

Proof. Set

$$\mathcal{T}a = \mathcal{T}(a) \qquad \mathcal{T}(O\,a) = \begin{cases} \texttt{<O b>} & \text{if } \mathcal{T}(a) = \{b\} \\ \texttt{<•>} & \text{otherwise} \end{cases} \qquad \mathcal{T}\bullet = \texttt{<•>}$$

□

**Proposition 4.4.** *There is a natural operation of type substitutions on message types such that* $\mathcal{T}\mu$ *is the least message type with the property*

$$[\![\mathcal{T}\mu]\!] \supseteq \{\, \mathcal{S}M \mid \vdash M : \mu \wedge \vdash \mathcal{S} : \mathcal{T} \,\}$$

Proof. To compute $\mathcal{T}\,\{C_1,\ldots,C_k\}*$, let $\mu_i = \mathcal{T}\,C_i$ for $1 \le i \le k$. If $\mu_i = \texttt{<>}$ for all $i$, then the result is also <>. Otherwise, the result is $\{C'_1,\ldots,C'_n\}*$, where the $C'_j$s are all capabilities that occur in any of the $\mu_i$'s, with duplicates removed (and in some canonical order).

To compute $\mathcal{T}\,\texttt{<}C_1.\cdots.C_k\texttt{>}$, let $\mu_i = \mathcal{T}\,C_i$ for $1 \le i \le k$, and then replace any $\{a\}$ among the $\mu_i$'s with <a>. (When a raw name is being concatenated with something else, its rawness disappears). If any $\mu_i$ has the form $\{\cdots\}*$, or if any $C$ appears in more than one $\mu_i$, then the result is $\mathcal{T}\,\{C_1,\ldots,C_k\}*$. Otherwise, each $\mu_i$ has the form $\texttt{<}\cdots\texttt{>}$. Concatenate all of the capability lists (in the order of the i's) and return <the concatenated list>.

Finally, $\mathcal{T}\,\{a\}$ is simply $\mathcal{T}(a)$. □

The $\supsetneq$ case of Proposition 4.4 arises, for example, for $[a \mapsto \texttt{<b>}]\texttt{<a.b>}$ which is $\{b\}*$ because <b.b> is not syntactically allowed.

**Definition 4.5 (Substitution for shape graphs).** *The following procedure defines an action of type substitutions on shape graphs. To construct $\mathcal{T}G$, first construct an intermediate graph $G_\varepsilon$ which can contain special null edges written $X \xrightarrow{\varepsilon} Y$. $G_\varepsilon$ contains contributions from each edge $Y_1 \xrightarrow{\pi} Y_2 \in G$:*

1. *When $\pi = a$ and $\mathcal{T}(a) = \{C_1,\ldots,C_k\}*$, choose a fresh node $Z$, and add to $G_\varepsilon$ the following edges:*

$$Y_1 \xrightarrow{\varepsilon} Z \xrightarrow{C_1} Z \xrightarrow{C_2} \cdots \xrightarrow{C_k} Z \xrightarrow{\varepsilon} Y_2$$

2. *When $\pi = a$ and $\mathcal{T}(a) = \texttt{<}C_1.\cdots.C_k\texttt{>}$, choose fresh nodes $Z_0$ through $Z_k$, and add to $G_\varepsilon$ the edges*

$$Y_1 \xrightarrow{\varepsilon} Z_0 \xrightarrow{C_1} Z_1 \xrightarrow{C_2} \cdots Z_k \xrightarrow{\varepsilon} Y_2$$

3. *When $\pi = a$ and $\mathcal{T}(a) = \{b\}$, add to $G_\varepsilon$ the edge $Y_1 \xrightarrow{b} Y_2$.*

4. *When $\pi = O\,a$, $\mathcal{T}(O\,a)$ will always have the form <C′>. Add to $G_\varepsilon$ the edge $Y_1 \xrightarrow{C'} Y_2$.*

5. *When $\pi = (a_1,\ldots,a_k)$, check that $\mathcal{T}\,a_i = \{a_i\}$ for all $i$, and then add the edge $Y_1 \xrightarrow{\pi} Y_2$ to $G_\varepsilon$. Otherwise, add $Y_1 \xrightarrow{\bullet} Y_2$.*

6. *When $\pi = \langle\mu_1,\ldots,\mu_k\rangle$, add to $G_\varepsilon$ the edge $Y_1 \xrightarrow{\langle\mathcal{T}\mu_1,\ldots,\mathcal{T}\mu_1\rangle} Y_2$.*

*Now set $\mathcal{T}G = \{\, X_k \xrightarrow{\pi} Y \mid (X_k \xrightarrow{\varepsilon} X_{k-1} \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} X_0 \xrightarrow{\pi} Y) \in G_\varepsilon, k \ge 0 \,\}$.*

The intermediate graph $G_\varepsilon$ can be given meaning if the rules of Definition 3.1 are supplemented by one for the special $\xrightarrow{\varepsilon}$ edges:

$$\frac{X \xrightarrow{\varepsilon} Y \qquad \vdash P : \langle Y \mid G\rangle}{\vdash P : \langle X \mid G\rangle}\ \textsf{Eps}$$

**Lemma 4.6.** $\vdash P : \langle X \mid \mathcal{T}G\rangle$ *if and only if* $\vdash P : \langle X \mid G_\varepsilon\rangle$

Proof. By direct manipulation of derivations. The "only if" direction is immediate. For the "if" direction, start by rewriting the derivation of $\vdash \mathsf{P} : \langle \mathsf{X} \mid \mathsf{G}_\varepsilon \rangle$ by distributing each use of the Eps rule over Seq, Nop, Par, and Bang to get them as far up in the tree as possible, and annihilate Eps'es that meet Null. Afterwards, all remaining Eps steps appear in neat stacks below Pfx steps, and a derivation of $\vdash \mathsf{P} : \langle \mathsf{X} \mid \mathcal{T}\mathsf{G} \rangle$ can be extracted directly. □

**Definition 4.7 (Substitution for shape predicates).** *Define the action on type substitutions on shape predicates by*

$$\mathcal{T}\langle \mathsf{X} \mid \mathsf{G} \rangle = \langle \mathsf{X} \mid \mathcal{T}\mathsf{G} \rangle$$

**Theorem 4.8.** *Assume that* $\vdash \mathsf{P} : \sigma$ *and* $\vdash \mathcal{S} : \mathcal{T}$*. Then* $\vdash \mathcal{S}\mathsf{P} : \mathcal{T}\sigma$*.*

Proof. First prove by induction on the derivation of $\vdash \mathsf{P} : \langle \mathsf{X} \mid \mathsf{G} \rangle$ that $\vdash \mathsf{P} : \langle \mathsf{X} \mid \mathsf{G} \rangle$ and $\vdash \mathcal{S} : \mathcal{T}$ imply $\vdash \mathcal{S}\,\mathsf{P} : \langle \mathsf{X} \mid \mathsf{G}_\varepsilon \rangle$. In the Pfx case, do a case analysis similar to the one in Definition 4.5.

Now, by Lemma 4.6, $\vdash \mathcal{S}\,\mathsf{P} : \langle \mathsf{X} \mid \mathcal{T}\mathsf{G} \rangle$. □

It would be convenient if one could think of $\mathcal{T}\sigma$ as the *least* shape predicate that makes Theorem 4.8 true. Unfortunately that is not the case. As a counterexample, set $\mathcal{T} = [\mathtt{a} \mapsto \{\mathtt{b}\}*]$ and $\sigma = \langle \mathtt{X0} \mid \mathtt{X0} \xrightarrow{\mathtt{a}} \mathtt{X1} \xrightarrow{\mathtt{c}} \mathtt{X2} \xrightarrow{\mathtt{a}} \mathtt{X3} \rangle$. Then

$$\mathcal{T}\sigma = \left\langle \mathtt{X0} \;\middle|\; \mathtt{X0} \overset{\overset{\mathtt{c}}{\frown}}{\xrightarrow{\mathtt{b}}} \mathtt{Y1} \xrightarrow{\mathtt{c}} \mathtt{X2} \xrightarrow{\mathtt{b}} \mathtt{Y3} \right\rangle \quad (\mathtt{Y1}\ \mathtt{b}\circlearrowright,\ \mathtt{Y3}\ \mathtt{b}\circlearrowright)$$

which allows any number of b's before a c and, independently, any number of b's after it. However, every term that can arise as $\mathcal{S}\,\mathsf{P}$ will have equally many b's on either side of the c, because the *same* sequence of b's get substituted for each a. Therefore the shape predicate

$$\left\langle \mathtt{X0} \;\middle|\; \mathtt{X0} \xrightarrow{\mathtt{c}} \mathtt{X2a} \overset{\overset{\mathtt{b}}{\frown}}{\mathtt{Y1}} \xrightarrow{\mathtt{c}} \mathtt{X2} \xrightarrow{\mathtt{b}} \mathtt{Y3} \right\rangle \quad (\mathtt{Y1}\ \mathtt{b}\circlearrowright,\ \mathtt{Y3}\ \mathtt{b}\circlearrowright)$$

would give a tighter approximation of the set of possible $\mathcal{S}\,\mathsf{P}$. This alternative special-cases the case where there are zero b's before the c, but it is clear that one could also construct shape predicates that similarly special-case terms with exactly one (two, three ...) b's before the c. In this way we can construct an infinite sequence of ever better approximations, which does not have any finite limit.

As a way around this infinite descent, we define:

**Definition 4.9.** *The shape graph* G *is* ***discrete*** *iff both of these hold:*

1. *For each capability* C *that is not* amb a *for some* a*, whenever* G *contains a chain* $\mathsf{X}_0 \xrightarrow{\mathsf{C}} \mathsf{X}_1 \xrightarrow{\mathsf{C}} \cdots \xrightarrow{\mathsf{C}} \mathsf{X}_k$ *of edges all decorated with* C *and any two of the* $\mathsf{X}_i$*'s are identical, then* $\mathsf{X}_0 = \mathsf{X}_1 = \cdots = \mathsf{X}_k$*.*

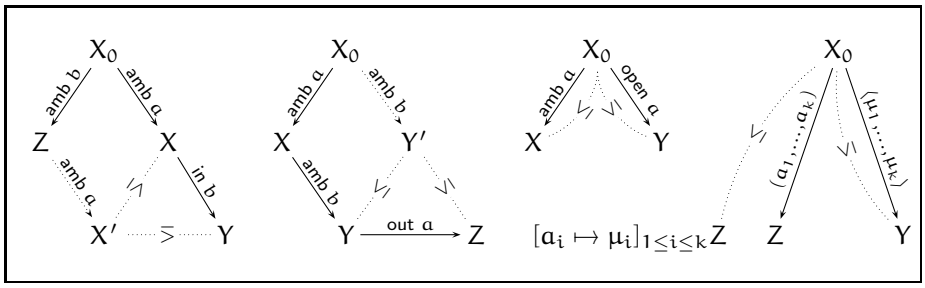


*Figure 4: Definition 5.2 in diagram form*

2. G *does not contain any message type of the shape* $\{C_1, \ldots, C_k\}*$ *such that one of the* $C_i$*'s is* amb a.

**Definition 4.10.** *A type substitution* $\mathcal{T}$ *is* ***discrete*** *iff there is no* b *such that* $\mathcal{T}(b)$ *is* $\{C_1, \ldots, C_k\}*$ *where one of the* $C_i$*'s is* amb a.

We can now formulate a partial inverse to Theorem 4.8:

**Theorem 4.11.** *Let* $\mathcal{T}$ *and* $\sigma$ *be given, and assume that* $\sigma'$ *satisfies all of the following:*

1. $\sigma'$ *is discrete.*
2. $\mathcal{T}$ *is discrete.*
3. *Whenever* $\vdash P : \sigma$ *and* $\vdash \mathcal{S} : \mathcal{T}$*, it holds that* $\vdash \mathcal{S}P : \sigma'$.

*Then* $\mathcal{T}\sigma \leq \sigma'$. *(Proof in Appendix A).*

Thus, if $\mathcal{T}$ is discrete, then $\mathcal{T}\sigma$ is a lower for every *discrete* shape predicates that describe all term substitution results. Note, however, that $\mathcal{T}\sigma$ is not in general *itself* discrete.

# 5 Shape predicates as types

## 5.1 Closed shape predicates

**Definition 5.1.** *The shape predicate* $\sigma$ *is* ***semantically closed*** *iff its meaning is closed under reduction, i.e., if* $\vdash P : \sigma$ *and* $P \hookrightarrow Q$ *imply* $\vdash Q : \sigma$.

This definition is intuitively appealing, but it is not immediately clear how to decide it. However, we have local rules that imply (and are under certain conditions equivalent to) semantic closure:

**Definition 5.2.** *The shape graph* G *is* ***locally closed*** *at* $X_0$ *iff*

1. $\{(X_0 \xrightarrow{\text{amb a}} X), (X \xrightarrow{\text{in b}} Y), (X_0 \xrightarrow{\text{amb b}} Z)\} \subseteq G$
   $\Rightarrow\ \exists X' : Z \xrightarrow{\text{amb a}} X' \in G\ \wedge\ \langle X \,|\, G\rangle \leq \langle X' \,|\, G\rangle\ \wedge\ \langle Y \,|\, G\rangle \leq \langle X' \,|\, G\rangle$,

2. $\{(X_0 \xrightarrow{\text{amb } a} X), (X \xrightarrow{\text{amb } b} Y), (Y \xrightarrow{\text{out } a} Z)\} \subseteq G$
   $\Rightarrow\ \exists Y' : X_0 \xrightarrow{\text{amb } b} Y' \in G\ \wedge\ \langle Y \mid G\rangle \leq \langle Y' \mid G\rangle\ \wedge\ \langle Z \mid G\rangle \leq \langle Y' \mid G\rangle$,

3. $\{(X_0 \xrightarrow{\text{amb } a} X), (X_0 \xrightarrow{\text{open } a} Y)\} \subseteq G$
   $\Rightarrow\ \langle X \mid G\rangle \leq \langle X_0 \mid G\rangle\ \wedge\ \langle Y \mid G\rangle \leq \langle X_0 \mid G\rangle$, *and*

4. $\{(X_0 \xrightarrow{\langle \mu_1,\ldots,\mu_k\rangle} Y), (X_0 \xrightarrow{(a_1,\ldots,a_k)} Z)\} \subseteq G$
   $\Rightarrow\ \langle Y \mid G\rangle \leq \langle X_0 \mid G\rangle\ \wedge\ [a_i \mapsto \mu_i]_{1\leq i\leq k}\langle Z \mid G\rangle \leq \langle X_0 \mid G\rangle$.

Figure 4 shows diagrams corresponding to each of the cases in this definition.

**Definition 5.3.** *Let* $\sigma = \langle X \mid G\rangle$ *be a shape predicate. The* ***active nodes*** *in* $\sigma$*, written* $\mathsf{active}(\sigma)$*, is the least set of node names such that*

$$\mathsf{active}(\sigma) = \{X\} \cup \{\, Z \mid \exists Y \in \mathsf{active}(\sigma) : \exists a : Y \xrightarrow{\text{amb } a} Z \in G \,\}.$$

**Definition 5.4.** *The shape predicate* $\langle X \mid G\rangle$ *is* ***syntactically closed*** *iff* $G$ *is locally closed at every* $X \in \mathsf{active}(\langle X \mid G\rangle)$.

**Definition 5.5.** *The shape graph* $G$ *is* ***trim*** *iff*

$$X \xrightarrow{\pi} Y \in G\ \wedge\ X \xrightarrow{\pi'} Z \in G\ \wedge\ \pi \leq \pi'\ \wedge\ \langle Y \mid G\rangle \leq \langle Z \mid G\rangle\ \Longrightarrow\ Y = Z.$$

Thus a trim graph is one where no edges can be taken away without changing its meaning.

**Lemma 5.6.** *Every shape graph* $G$ *has a trim equivalent subgraph: For any* $G$ *there exists a trim* $G' \subseteq G$ *such that* $G \approx G'$.

Proof. If the graph is *not* trim, then there is an $X \xrightarrow{\pi} Y$ edge that can be removed without changing the meaning of it. That makes the graph smaller; now proceed by induction on the size of $G$. □

Note that the procedure sketched in the proof is nondeterministic. For example,

$$G = \{(\texttt{X0} \xrightarrow{\texttt{a}} \texttt{X1}), (\texttt{X0} \xrightarrow{\texttt{b}} \texttt{X1}), (\texttt{X0} \xrightarrow{\texttt{b}} \texttt{X2})\}$$

has two different (non-isomorphic) trim equivalents, depending on which of the `b` edges one chooses to remove.

**Theorem 5.7.** *Let* $\sigma$ *be a shape predicate.*

1. *If* $\sigma$ *is syntactically closed, then it is semantically closed.*
2. *If* $\sigma$ *is trim and discrete, then the reverse implication also holds:* $\sigma$ *is syntactically closed if and only if it is semantically closed.*

*(Proof in Appendix B).*

The conditions in Theorem 5.7(2) are necessary. A shape predicate that is semantically closed but neither discrete nor syntactically closed is

$$\mathsf{letrec}\ X = ((\mathtt{a}).\mathtt{a}.\mathtt{a}.\mathtt{c}.0 \mid \langle\{\mathtt{b}\}*\rangle.0 \mid \mathtt{b}.\mathtt{b}.X \mid \mathtt{c}.0)\ \mathsf{in}\ X.$$

One that is semantically closed but neither trim nor syntactically closed is

```
a[in b.0] | b[a[in b.0]] | b[0].
```

## 5.2 Types

**Definition 5.8.** *A **type** $\tau$ is a syntactically closed shape predicate. Given a type $\tau$, the term $P$ has type $\tau$ iff $\vdash P : \tau$.*

This notion of types has the basic properties expected of any type system: It enjoys subject reduction (Theorem 5.7(1)), it can be effectively decided whether a given term has a given type (Proposition 3.5), and types can be distinguished from non-types (using Proposition 3.16).

An algorithm to compute precise types (such as the one presented by Makholm and Wells [2004]) can be used to approximate various properties of a term's computational behaviour:

- If $P$ has the type $\sigma = \langle X \mid G \rangle$ and $G$ contains no edge $Y \xrightarrow{\bullet} Z$ with $Y \in \mathsf{active}(\sigma)$, then executing $P$ will never *execute* a malformed substitution result such as $[a \mapsto M.N](\mathsf{in}\ a)$.
- If $P$ has the type $\langle X \mid G \rangle$ and $G$ contains no edge $Y \xrightarrow{\bullet} Z$, then executing $P$ will never create a malformed substitution result.
- Any **security policy** can be checked if it can be stated as a condition on configurations that must not arise. For example, the policy "no ambient $a$ must ever directly contain an ambient named $b$" is satisfied by $P$ if it has a type $\langle X \mid G \rangle$ such that $G$ does not contain a sequence $X_1 \xrightarrow{\mathsf{amb}\ a} X_2 \xrightarrow{\mathsf{amb}\ b} X_3$.

**Proposition 5.9.** *Every term $P$ has a type – although the type may contain $\bullet$ and thus not prove that the term "cannot go wrong".*

Proof. Let $\mathbf{A}$ be the (finite) set of names mentioned in $P$, and let $k$ be the maximal arity of communication actions in $P$. Now let

$$\begin{aligned}
\{C_1, \ldots, C_m\} &= \mathbf{A} \cup \{\, O\ a \mid O \in \boxed{O}, a \in \mathbf{A} \,\} \cup \{\bullet\} \\
\mathbf{M} &= \{\, \{a\} \mid a \in \mathbf{A} \,\} \cup \{\, \{C_1, \ldots, C_m\}* \,\} \\
\Pi &= \{C_1, \ldots, C_m\} \cup \\
&\quad \{\, (a_1, \ldots, a_n) \mid a_i \in \mathbf{A}, n \leq k \,\} \cup \\
&\quad \{\, \langle \mu_1, \ldots, \mu_n \rangle \mid \mu_i \in \mathbf{M}, n \leq k \,\} \\
\tau &= \langle X_0 \mid \{\, X_0 \xrightarrow{\pi} X_0 \mid \pi \in \Pi \,\} \rangle
\end{aligned}$$

It is easy to see that $\tau$ is matched by every process term that can be built using only the names in $\mathbf{A}$ with communication arities up to $k$. Therefore $\mathcal{T}\,\tau \leq \tau$ whenever the image of $\mathbf{A}$ by $\mathcal{T}$ contains only names in $\mathbf{A}$, because every name in $\mathcal{T}\,\tau$ comes from either $\mathcal{T}$ or $\tau$.

It is now straightforward to verify that $\tau$ is syntactically closed. Thus $\tau$ is indeed a type, and clearly $\vdash P : \tau$, as required. □

Our notion of types is very expressive — it *allows* a very fine-grained approximation to important questions. However, it is not known whether principal types

always exist; we have neither proved nor disproved this. Thus, we now define a syntactically restricted type system for which we *do* prove that principal types exist.

## 5.3 Modest types; existence of principal types

**Definition 5.10.** *Define the relation $=_{(\leq)}$ on prefix types as the least equivalence relation that contains $\leq$.*

$=_{(\leq)}$ is close to being the identity: The only pairs of *different* prefix types that are related are $\langle \mu_1, \ldots, \mu_n \rangle$ and $\langle \mu'_1, \ldots, \mu'_n \rangle$ (with the same arity $n$), where for each $i$ it holds that either $\mu_i = \mu'_i = \{a\}$ or neither of $\mu_i$ and $\mu'_i$ has the form $\{a\}$.

**Definition 5.11.** *Define the **stratification** function* $\mathbf{S}$ *by*

$$\mathbf{S}((\vec{a})) = \mathbf{S}(\langle \vec{\mu} \rangle) = 3 \qquad \mathbf{S}(\mathsf{amb}\ a) = 2 \qquad \mathbf{S}(C) = 1 \text{ when } C \neq \mathsf{amb}\ a$$

Obviously, $\pi =_{(\leq)} \pi'$ implies $\mathbf{S}(\pi) = \mathbf{S}(\pi')$.

**Definition 5.12.** *The shape graph* $G$ *is **modest** iff for each $\pi$, one of the following conditions hold:*

1. ***Finite depth.*** *There is a number $n_\pi$ such that whenever $G$ contains a chain $X_0 \xrightarrow{\pi_1} X_1 \xrightarrow{\pi_2} \cdots \xrightarrow{\pi_k} X_k$ with every $\mathbf{S}(\pi_i) \leq \mathbf{S}(\pi)$, there are at most $n_\pi$ different $i$'s such that $\pi_i =_{(\leq)} \pi$.*
2. ***Monomorphic recursion.*** *Whenever $G$ contains a chain $X_0 \xrightarrow{\pi_1} X_1 \xrightarrow{\pi_2} \cdots \xrightarrow{\pi_k} X_k$ with every $\mathbf{S}(\pi_i) \leq \mathbf{S}(\pi)$ and $\pi_1 =_{(\leq)} \pi =_{(\leq)} \pi_k$, then $X_1 = X_k$.*

Modesty is a rather technical concept; the main property that it has to satisfy is stated as Proposition 5.15 below. Here are some comments that may help getting an intuition about what modesty means. The basic idea is to restrict *cycles* so that they cannot be arbitrarily long without mentioning arbitrarily many different capabilities. A graph without any cycles will always be modest, because it satisfies finite depth for all $\pi$.

However, as remarked in Section 3.1, acyclic shape predicates are not enough to type all interesting terms. We therefore allow cycles, as long as they satisfy "monomorphic recursion". This condition says that if we need cycles containing some prefix — say, the ambient boundary `amb a` — then two nested ambients of that name *must* match the same node in the graph. This means that `a[(b).in b.0 | !a[⟨ε⟩.0 | !in a.0]]` cannot be typed with a $\bullet$-free modest type, because the monomorphic recursion requirement forces the type inside the two `a` ambients to be the same node in the graph. Therefore the type system wrongly thinks that $\langle \varepsilon \rangle$ may communicate with `(b).in b.0`.

The stratification function $\mathbf{S}$ eases the modesty requirements. It says that a cycle only "counts as a cycle" for the prefix type in it at the highest stratum. Thus the chain $X_1 \xrightarrow{\mathsf{out}\ a} X_2 \xrightarrow{\mathsf{amb}\ b} X_3 \xrightarrow{\mathsf{out}\ a} X_4$ does not violate monomorphic

recursion for $\mathtt{out}\ a$. Because ambient boundaries are in their own stratum, there can be local $X \xrightarrow{C} X$ edges within each ambient without forcing all nodes in the graph to collapse.

It is always possible to satisfy finite depth for stratum-3 prefix types, because none of the reduction rules increase communication action nesting depth. Therefore, one needs only consider shape graphs consisting of clusters of capability-marked edges, linked together by stratum-3 edges in a *tree*.

The flexibility offered by stratification is restricted because one must choose *globally* between finite depth and monomorphic recursion for each $\pi$, rather than in each isolated cluster of stratum-$n$-and-lower edges. This restriction is not intuitive, but is needed for preservation of modesty in Proposition 5.16 below.

Allowing only modest *and* discrete types yields **principal typings** (defined by Wells [2002]):

**Theorem 5.13.** *For every term* $P$ *which has at least one modest discrete type, there is a modest discrete type* $\tau$ *that is minimal among* $P$*'s modest discrete types.*

For the sake of proving Theorem 5.13, we will temporarily consider *infinite shape graphs* which may contain a (generally uncountable) infinity of edges between an infinity of node names. Infinite shape graphs can, of course, be used in infinite shape predicates, and matching between infinite predicates and (ordinary finite) terms can be defined by the same rules as for finite shape predicates.

**Definition 5.14.** *An (infinite) shape graph is* ***finitary*** *if it contains only finitely many different prefix types.*

Obviously any finite shape graph is also finitary.

The proofs of the following three propositions are somewhat involved and can be found in Appendix C. (Proposition 5.15 is the key step in the proof of Proposition 5.17).

**Proposition 5.15.** *Let* $G$ *be a finitary modest shape graph, and define the equivalence relation* $\sim_G$ *by*

$$X \sim_G Y \iff [\![\langle X \mid G \rangle]\!] = [\![\langle Y \mid G \rangle]\!]$$

*There are only finitely many equivalence classes with respect to* $\sim_G$.

**Proposition 5.16.** *Let* $(\sigma_i)_{i \in I}$ *be a (finite or infinite but nonempty) family of finite, modest, discrete shape predicates. Then there exists a finitary, modest, discrete* $\sigma$ *such that*

$$[\![\sigma]\!] = \bigcap_{i \in I} [\![\sigma_i]\!]$$

**Proposition 5.17.** *Let* $\sigma_\infty$ *be a finitary, modest, discrete shape predicate. There exists a finite, modest, discrete, trim shape predicate* $\sigma'$ *such that* $[\![\sigma']\!] = [\![\sigma_\infty]\!]$.

**Proof of Theorem 5.13.** Let **T** be the set of (finite) modest discrete types of the given term P; it is nonempty by assumption. Apply Proposition 5.16 to **T**; this produces a finitary (but in general infinite), modest, discrete $\sigma_\infty$ such that

$$[\![\sigma_\infty]\!] = \bigcap_{\tau \in \mathbf{T}} [\![\tau]\!]$$

Now apply Proposition 5.17 to $\sigma_\infty$; that produces a *finite*, modest, discrete, trim $\sigma$ such that

$$[\![\sigma]\!] = [\![\sigma_\infty]\!] = \bigcap_{\tau \in \mathbf{T}} [\![\tau]\!]$$

By Theorem 5.7, each $\tau \in \mathbf{T}$ is semantically closed, and it is now obvious from the definition of semantic closure that $\sigma$ is also semantically closed. Another application of Theorem 5.7 tells us that $\sigma$ is syntactically closed and, therefore, in fact a type. It is clearly the *least* type for P. □

It is evident that this proof is non-constructive (it assumes that the entire set of possible types are given) and does not point to an effective procedure for *finding* a principal type. Makholm and Wells [2004] have defined (and implemented) a practical type inference algorithm for a yet more restricted version of PolyA, but its principality properties are not yet well understood.

Requiring discreteness of types loses Proposition 5.9: There exist terms having no discrete type. One example is $\langle \mathtt{amb\ a} \rangle.0 \mid\ !(\mathtt{b}).\langle \mathtt{b.b} \rangle.0$, which constructs messages that are arbitrarily long sequences of ambient-construction operations. To type this, we need the message type $\{\mathtt{amb\ a}\}*$, which is, however, explicitly forbidden by the definition of discreteness. However all ($\nu$-free) terms of the original ambient calculus have types:

**Proposition 5.18.** *Any term* P *that does not contain* amb a *inside* $\langle \vec{M} \rangle$ *has a modest discrete type, and so also a principal such.*

Proof. Similar to Proposition 5.9, except that capabilities of the form amb a are excluded from the construction of **M**. □

# 6 Name restriction

So far we have ignored the *name restriction* of the original ambient calculus. Now we present a simple scheme for putting it back into the type system. In later work it may be possible to combine PolyA with more advanced treatments of name restriction, such as the "abstract names" of Lhoussaine and Sassone [2004].

The syntax and semantics of name restriction are hopefully well-known:

$$P ::= \cdots \mid \nu a.P \qquad\qquad \frac{P \hookrightarrow Q}{\nu a.P \hookrightarrow \nu a.Q}$$

$$\frac{P \equiv Q}{\nu a.P \equiv \nu a.Q} \qquad \frac{a \text{ not free in } P}{P \mid \nu a.Q \equiv \nu a.(P \mid Q)}$$

$$\frac{}{\nu a.\nu b.P \equiv \nu b.\nu a.P} \qquad \frac{a \neq b}{a[\nu b.P] \equiv \nu b.a[P]} \qquad \frac{}{\nu a.0 \equiv 0}$$

In contrast to what we did for $(\vec{a})$ actions, we view $\alpha$-equivalence of $\nu$ binding as a primitive identity – the terms $\nu a.P$ and $\nu b.[a \mapsto b]P$ are considered *identical* (if $b$ is not free in $P$). For simplicity, we require that the body of a $\nu$-binding does not contain a $(\vec{a})$ prefix that re-bind the $\nu$-bound name.

The main problem with name restriction is that its position in the ambient hierarchy is not fixed, so we cannot treat is like just another prefix. Furthermore, a name restriction can be replicated by a `!` and extrude to encompass its own original, by the equivalence

$$!\nu a.P \;\equiv\; \nu a.P \mid !\nu a.P = \nu b.[a \mapsto b]P \mid !\nu a.P \;\equiv\; \nu b.([a \mapsto b]P \mid !\nu a.P)$$

This pattern is ubiquitous in applications of the ambient calculus; it shows that we need to support name restriction in such a way that a type does not place an upper limit on the number of different $\nu$-bound names that can be in scope in any part of the term – otherwise we would lose the fundamental Theorem 3.2. This means that we need to let the *same* prefix types in the shape predicate match several different names in the actual term if they are bound by (perhaps nested) name restrictions.

(Note in passing that it is a fundamental premise of our type system that it ignores `!`'s, so it cannot distinguish between $!\nu a.P$ and $\nu a.!P$. Therefore it is intrinsically impossible for it to prove properties that depend on different copies of $P$ having distinct private names in the former case).

Handling $\alpha$-convertible binders directly in tree automata such as shape predicates is known to be hard [Glew 2002]; instead our idea is to omit the binders from the types entirely. A first sketch of a typing rule for name restriction is:

$$\frac{\vdash [a \mapsto b]P : \sigma}{\vdash \nu a.P : \sigma}$$

where $b$ is arbitrary! This lets us choose the same $b$ for several $\nu$ bindings in the term, such that they can match the same part of the shape predicate. However, this rule is a little too liberal to work – it leaves too little trace of what happened. For example, in order to type

$$\nu a.a[0]$$

we would need to choose a $b$ to use in the type, but no matter which $b$ we choose, we will end up with a type that also describes $b[0]$ *without* the name restriction. Clearly no type with this property can be principal for $\nu a.a[0]$, so we lose the principality property.

We can fix this by requiring that $b$ is chosen from a set $B$ that comes with the shape predicate, and that these names can be used *only* to match $\nu$-bound names in the term we are typing – they cannot be allowed to appear free in it. We do this by the following slightly indirect definition:

**Definition 6.1.** *Let the variable letter* $B$ *range over nonempty finite sets of names, and define the relation* $B \vdash P \curvearrowright P'$ *by*

$$\frac{B \vdash P \curvearrowright P'}{B \vdash p.P \curvearrowright p.P'} \qquad \frac{B \vdash P \curvearrowright P}{B \vdash !P \curvearrowright !P'} \qquad \frac{B \vdash P \curvearrowright P' \qquad B \vdash Q \curvearrowright Q'}{B \vdash P \mid Q \curvearrowright P' \mid Q'}$$

$$\frac{}{B \vdash 0 \curvearrowright 0} \qquad \frac{b \in B \qquad B \vdash [a \mapsto b]P \curvearrowright P'}{B \vdash \nu a.P \curvearrowright P'}$$

Intuitively $B \vdash P \curvearrowright P'$ works by choosing a $b \in B$ for each $\nu$-bound variable in $P$, and then deleting all of the $\nu$'s to get $P'$.

**Definition 6.2.** *A* $\nu$***-aware shape predicate*** *is a pair of a* $B$ *and a shape predicate* $\sigma$*, written* $\sigma/B$*. Its* ***meaning*** $[\![\sigma/B]\!]$ *is the set of all terms* $P$ *such that* $P$ *contains no* $b \in B$ *(except bound by* $\nu$*) and there is a* $P'$ *such that* $B \vdash P \curvearrowright P'$ *and* $\vdash P' : \sigma$*.*

Appendix D proves the following key properties of $\nu$-aware shape predicates:

**Theorem 6.3.** *The following all hold:*

1. $[\![\sigma/B]\!]$ *is closed under* $\equiv$*.*
2. *If* $\sigma \leq \sigma'$*, then* $[\![\sigma/B]\!] \subseteq [\![\sigma'/B]\!]$*.*
3. *If* $\sigma$ *is syntactically closed, then* $[\![\sigma/B]\!]$ *is closed under* $\hookrightarrow$*.*
4. *The system of discrete modest* $\nu$*-aware types enjoys a principal type property: For each* $P$*, if*

$$\{\, \tau/B \mid \tau \text{ is discrete and modest, and } P \in [\![\tau/B]\!] \,\}$$

*is nonempty, then it has a least element.*

# 7 Extended and modified ambient calculi

Our framework is strong enough to handle many ambient calculus variants with different reduction rules. In most cases, PolyA can be extended to deal with such variation simply by adjusting Definition 5.2 with conditions systematically derived from the changed or new reduction rules. If this is done correctly and the new or changed rules are straightforward rewriting steps, then it is simple to construct new cases for the proof of Theorem 5.7. The rest of our theory will then carry through unchanged, including the existence of principal types.

We illustrate this principle with examples of such extensions.

**Boxed Ambients** [Bugliesi et al. 2001] removes the open capability; instead processes can communicate across ambient boundaries with directional communication actions:

$$\text{Prefixes: } p ::= M \mid \langle \vec{M} \rangle^{\uparrow} \mid \langle \vec{M} \rangle^{\star} \mid \langle \vec{M} \rangle^{\downarrow a} \mid (\vec{a})^{\uparrow} \mid (\vec{a})^{\star} \mid (\vec{a})^{\downarrow a}$$

There are corresponding reduction rules such as:

$$\overline{\langle \vec{M} \rangle^{\downarrow b}.P \mid b[Q \mid (\vec{a})^{\star}.R] \hookrightarrow P \mid b[Q \mid [a_i \mapsto M_i]_i R]}$$

Our prefix type syntax is easily extended to include the new actions. The new reduction rules can be used to derive local closure conditions such as:

$$\{(X_0 \xrightarrow{\langle \mu_1,\ldots,\mu_k \rangle^{\downarrow b}} X), (X_0 \xrightarrow{\text{amb } b} Y), (Y \xrightarrow{(a_1,\ldots,a_k)^{\star}} Z)\} \subseteq G$$
$$\Rightarrow \langle X \mid G \rangle \leq \langle X_0 \mid G \rangle \wedge [a_i \mapsto \mu_i]_{1 \leq i \leq k} \langle Z \mid G \rangle \leq \langle Y \mid G \rangle$$

**Safe Ambients** [Levi and Sangiorgi 2000] introduces *co-capabilities* (also added to BA by Bugliesi et al. [2002]), where both interaction parties need a capability. This can improve analysis precision and avoid unwanted behaviours. The reduction rules are amended to require this, e.g.:

$$\overline{a[\overline{\text{open}}\ a.P \mid Q] \mid \text{open } a.R \hookrightarrow P \mid Q \mid R}$$

It is straightforward to extend PolyA to systems with co-capabilities. For example, condition 3 of Definition 5.2 would be replaced by:

$$\{(X_0 \xrightarrow{\text{amb } a} X), (X_0 \xrightarrow{\text{open } a} Y), (X \xrightarrow{\overline{\text{open}}\ a} Z)\} \subseteq G$$
$$\Rightarrow \langle X \mid G \rangle \leq \langle X_0 \mid G \rangle \wedge \langle Y \mid G \rangle \leq \langle X_0 \mid G \rangle \wedge \langle Z \mid G \rangle \leq \langle X_0 \mid G \rangle.$$

The $\mathbf{M^3}$ calculus [Coppo et al. 2003] introduces a new method of inter-ambient communication; a new capability to can move a process into a neighbour ambient:

$$\overline{a[P \mid \text{to } b.Q] \mid b[R] \hookrightarrow a[P] \mid b[Q \mid R]}$$

This, too, is easily expressed as a closure condition:

$$\{(X_0 \xrightarrow{\text{amb } b} X), (X_0 \xrightarrow{\text{amb } a} Y), (Y \xrightarrow{\text{to } b} Z)\} \subseteq G$$
$$\Rightarrow \langle Z \mid G \rangle \leq \langle X \mid G \rangle$$

# References


Amtoft, T., Kfoury, A. J., and Pericas-Geertsen, S. M. (2001). What are polymorphically-typed ambients? In Sands, D., editor, *ESOP 2001, Genova*, volume 2028 of *LNCS*, pages 206–220. Springer-Verlag. An extended version appears as Technical Report BUCS-TR-2000-021, Comp.Sci. Department, Boston University, 2000.

Amtoft, T., Kfoury, A. J., and Pericas-Geertsen, S. M. (2002). Orderly communication in the ambient calculus. *Computer Languages*, 28:29–60.

Amtoft, T., Makholm, H., and Wells, J. B. (2004). PolyA: True type polymorphism for Mobile Ambients. In *Theoretical Computer Science: 3rd IFIP Int'l Conf.* Kluwer Academic Publishers.

Bugliesi, M., Castagna, G., and Crafa, S. (2001). Boxed ambients. In *4th International Conference on Theoretical Aspects of Computer Science (TACS'01)*, volume 2215 of *LNCS*, pages 38–63. Springer-Verlag.

Bugliesi, M., Crafa, S., Merro, M., and Sassone, V. (2002). Communication interference in mobile boxed ambients. In *FST & TCS 2002*.

Cardelli, L., Ghelli, G., and Gordon, A. D. (1999). Mobility types for mobile ambients. In Wiedermann, J., van Emde Boas, P., and Nielsen, M., editors, *ICALP'99*, volume 1644 of *LNCS*, pages 230–239. Springer-Verlag. Extended version appears as Microsoft Research Technical Report MSR-TR-99-32, 1999.

Cardelli, L. and Gordon, A. D. (1998). Mobile ambients. In Nivat, M., editor, *FoSSaCS'98*, volume 1378 of *LNCS*, pages 140–155. Springer-Verlag.

Cardelli, L. and Gordon, A. D. (1999). Types for mobile ambients. In *POPL'99, San Antonio, Texas*, pages 79–92. ACM Press.

Cardelli, L. and Wegner, P. (1985). On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4):471–522.

Coppo, M. and Dezani-Ciancaglini, M. (1980). An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Logic*, 21(4):685–693.

Coppo, M. and Dezani-Ciancaglini, M. (2002). A fully abstract model for higher-order mobile ambients. In *VMCAI 2002*, volume 2294 of *LNCS*, pages 255–271.

Coppo, M., Dezani-Ciancaglini, M., Giovannetti, E., and Salvo, I. (2003). M3: Mobility types for mobile processes in mobile ambients. In *CATS 2003*, volume 78 of *ENTCS*.

Glew, N. (2002). A theory of second-order trees. In *ESOP 2002*, volume 2305 of *LNCS*, pages 147–161. Springer-Verlag.

Levi, F. and Maffeis, S. (2001). An abstract interpretation framework for analysing mobile ambients. In *SAS'01*, volume 2126 of *LNCS*, pages 395–411. Springer-Verlag.

Levi, F. and Sangiorgi, D. (2000). Controlling interference in ambients. In *POPL'00, Boston, Massachusetts*, pages 352–364. ACM Press.

Lhoussaine, C. and Sassone, V. (2004). A dependently typed ambient calculus. In *Programming Languages & Systems, 13th European Symp. Programming*, volume 2986 of *LNCS*. Springer-Verlag.

Maffeis, S. (2004). Sequence types for the π-calculus. In *Intersection Types and Related Systems*. To appear in ENTCS.

Makholm, H. and Wells, J. B. (2004). Type inference for PolyA. Technical Report HW-MACS-TR-0013, Heriot-Watt Univ., School of Math. & Comput. Sci.

Milner, R. (1999). *Communicating and Mobile Systems: The π-Calculus*. Cambridge Press.

Nielson, F., Nielson, H. R., and Sagiv, M. (2000). A Kleene analysis of mobile ambients. In *Programming Languages & Systems, 9th European Symp. Programming*, volume 1782 of *LNCS*, pages 305–319. Springer-Verlag.

Nielson, H. R. and Nielson, F. (2000). Shape analysis for mobile ambients. In *POPL'00, Boston, Massachusetts*, pages 142–154. ACM Press. A revised and extended version has appeared in Nordic Journal of Computing, 8:233–275, 2001.

Sander, G. (1994). Graph layout through the VCG tool. In Tamassia, R. and Tollis, I. G., editors, *Graph Drawing: DIMACS International Workshop, GD '94*, volume 894 of *LNCS*, pages 194–205. Springer-Verlag.

Teller, D., Zimmer, P., and Hirschkoff, D. (2002). Using ambients to control resources. In *CONCUR'02*, volume 2421 of *LNCS*, pages 288–303. Springer-Verlag.

Turner, D. N. (1995). *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, University of Edinburgh. Report no ECS-LFCS-96-345.

Wells, J. B. (2002). The essence of principal typings. In *Proc. 29th Int'l Coll. Automata, Languages, and Programming*, volume 2380 of *LNCS*, pages 913–925. Springer-Verlag.

Zimmer, P. (2000). Subtyping and typing algorithms for mobile ambients. In *FOSSACS 2000, Berlin*, volume 1784 of *LNCS*, pages 375–390. Springer-Verlag.

# A Proof of Theorem 4.11

Assume that $\sigma = \langle X \mid G \rangle$, $\mathcal{T}$, and $\sigma' = \langle X' \mid G' \rangle$ are as in the theorem, and define $\mathcal{S}$ in the rest of this appendix by

$\mathcal{S}(a) = b$ when $\mathcal{T}(a) = \{b\}$

$\mathcal{S}(a) = C_1.\cdots.C_k.\varepsilon$ when $\mathcal{T}(a) = <C_1.\cdots.C_k>$.

$\mathcal{S}(a) = (C_1.\cdots.C_1).\cdots.(C_k.\cdots.C_k)$ when $\mathcal{T}(a) = \{C_1, \ldots, C_k\}*$. There are $|G'|$ (but at least 2) copies of each $C_i$. Because $\mathcal{T}$ is discrete, none of the $C_i$'s have the form amb b.

It is clear that $\vdash \mathcal{S} : \mathcal{T}$.

**Lemma A.1.** *If* $\vdash \mathcal{S}\lceil\mu\rceil : \mu'$ *(where* $\lceil\mu\rceil$ *is as in Lemma 3.9), then* $\mathcal{T}\mu \leq \mu'$.

Proof. By case analysis on $\mu$.

1. $\mu = \{C_1, \cdots, C_k\}*$. Then $\mathcal{S}\lceil\mu\rceil = \mathcal{S}C_1.\mathcal{S}C_1.\cdots.\mathcal{S}C_k.\mathcal{S}C_k$. If all $\mathcal{S}C_i$ are $\varepsilon$, then all $C_i$'s must be names $a_i$ with $\mathcal{T}(a_i) = <>$. Therefore $\mathcal{T}\mu = <>$, and $\mathcal{T}\mu \leq \mu'$ follows immediately.

   Otherwise, at least one $\mathcal{S}C_i$ contains a capability, which then occurs at least twice in $\mathcal{S}\lceil\mu\rceil$. Therefore $\mu'$ must have the form $\{\vec{C}\}*$ and contain every capability that appears in any $\mathcal{S}C_i$. Those are exactly the capabilities that appear in $\mathcal{T}\mu$, so again $\mathcal{T}\mu \leq \mu'$ follows.

2. $\mu = <C_1.\cdots.C_k>$. Then $\mathcal{S}\lceil\mu\rceil = \mathcal{S}C_1.\cdots.\mathcal{S}C_k.\varepsilon$, so $\mu'$ must must have one of the forms $<\cdots>$ and $\{\cdots\}*$.

   Assume that $\mu'$ is $<\vec{C}>$. Then none of the $\mathcal{S}C_i$ can contain the same message twice, so no $C_i$ can be a $a$ with $\mathcal{T}(a) = \{\cdots\}*$. Therefore no $\mathcal{T}\,C_i$ can be $\{\cdots\}*$ either. That means that $\mathcal{T}\,\mu$ will have the shape $<\cdots>$ unless there is some duplicated capability in $\mathcal{T}\,C_1, \ldots\, \mathcal{T}\,C_k$. But the capabilities in each $\mathcal{T}\,C_i$ are exactly the same as the ones in $\mathcal{S}\,C_i$. Therefore there cannot be any such duplication, so $\mathcal{T}\,\mu$ does indeed have the shape $<\vec{C}'>$. In fact $\vec{C}'$ must contain exactly the same capabilities, and in the same order as $\vec{C}$, so $\mathcal{T}\,\mu = \mu'$.

   The case that $\mu'$ has the form $\{\vec{C}\}*$ remains. In that case, the list $\vec{C}$ contains every capability that appears in any $\mathcal{S}C_i$. Those are exactly the capabilities that appear in $\mathcal{T}\mu$, so again $\mathcal{T}\mu \leq \mu'$ follows.

3. $\mu = \{a\}$. Then $\mathcal{S}\lceil\mu\rceil = \mathcal{S}(a)$ and $\mathcal{T}\mu = \mathcal{T}(a)$. In each of the cases in the definition of $\mathcal{S}(a)$ it is clear (as in Lemma 3.9) that $\vdash \mathcal{S}(a) : \mu'$ implies $\mathcal{T}(a) \leq \mu'$. □

Call a node $Y$ **pristine** if it existed in the original $G$ (as opposed to having been chosen fresh during the construction of $G_\varepsilon$), and consider the operation of splitting a derivation $\vdash P : \langle Y_0 \mid G_\varepsilon \rangle$ (where $Y_0$ is pristine) into a tree of derivation **fragments** such that there is a fragment boundary at each place where two inference steps are connected[4] by a judgement of of the form $\vdash Q : \langle Y \mid G_\varepsilon \rangle$ where $Y$ is pristine. A generic fragment looks like

$$\dfrac{\vdash R_1 : \langle Y_2 \mid G_\varepsilon \rangle \quad \cdots \quad \vdash R_n : \langle Y_2 \mid G_\varepsilon \rangle}{\vdots \qquad\qquad \vdots \qquad \vdots}\ \textit{(rule name)}$$
$$\vdash Q : \langle Y_1 \mid G_\varepsilon \rangle$$

where the dotted part stands for zero or more inference steps apart from the one that concluded $\vdash Q : \langle Y_1 \mid G_\varepsilon \rangle$. There may be zero or more $R_i$'s at the top of the fragment. It happens that they all have the same shape-predicate part $\langle Y_2 \mid G_\varepsilon \rangle$, irrespective of whether the bottommost rule in the fragment is Pfx or Eps (in which case it can be seen from the construction of $G_\varepsilon$ that only one $Y_2$ will be reachable without passing through another) or not (in which case the fragment consists of just one inference and $Y_2 = Y_1$).

**Lemma A.2.** *Assume $\vdash P : \langle Y \mid G_\varepsilon \rangle$ such that $Y$ is pristine. There exists a term $P^* \in [\![\langle Y \mid G \rangle]\!]$ such that for all $Y'$, $\vdash \mathcal{S}\, P^* : \langle Y' \mid G' \rangle$ implies $\vdash P : \langle Y' \mid G' \rangle$.*

Proof. Split the derivation of $\vdash P : \langle Y \mid G_\varepsilon \rangle$ into fragments, and use induction on the tree of fragments. Let the premises at the top of the bottommost fragment be $R_1$ through $R_n$, and use the induction hypothesis to produce $R_1^*$ through $R_n^*$.

If the fragment consists of a single non-Pfx, non-Eps rule, simply connect the $R_i^*$'s from the induction hypothesis to form $P^*$ in the same way the $R_i$'s form $P$.

The interesting case is where the fragment ends with a Pfx or Eps rule. In each case, the rule depends on edges in $G_\varepsilon$ that were added due to a particular edge $Y_1 \xrightarrow{\pi} Y_2 \in G$. Set $P^* = \lceil \pi \rceil .(R_1^* \mid \cdots \mid R_n^*)$, where $\lceil \pi \rceil$ is as in Corollary 3.10. Then it is clear that $\vdash P^* : \langle Y_1 \mid G \rangle$. To see that $\vdash \mathcal{S}\, P^* : \langle Y' \mid G' \rangle$ implies $\vdash P : \langle Y' \mid G' \rangle$, divide into cases according to which case of the construction of $G_\varepsilon$ was used to handle the edge used in the last rule.

1. $\pi = a$ and $\mathcal{T}(a) = \{C_1, \ldots, C_k\}*$. Because $\lceil a \rceil = a$, we have $\mathcal{S}\, P^* = \mathcal{S}(a).(\cdots \mid \mathcal{S}\, R_i^* \mid \cdots)$. If $\vdash \mathcal{S}\, P^* : \langle Y' \mid G' \rangle$ for some $Y'$, then $G'$ must contain a chain

   $$Y' = Y_0^1 \xrightarrow{C_1} Y_1^1 \xrightarrow{C_1} \cdots \xrightarrow{C_1} Y_{|G'|}^1 = Y_0^2 \xrightarrow{C_2} Y_1^2 \xrightarrow{C_2} \cdots \xrightarrow{C_k} Y_{|G'|}^k$$

   Since, for each $i$ the $C_i$ subchain contains more than $|G'|$ $Y_j^i$'s, some of them must be identical. Because $G'$ is discrete, this means that every $Y_j^i = Y'$. We get $\vdash \mathcal{S}\, R_i^* : \langle Y' \mid G' \rangle$ and so $\vdash R_i : \langle Y' \mid G' \rangle$.

   To derive $\vdash P : \langle Y' \mid G' \rangle$, take the derivation fragment for $\vdash P : \langle Y' \mid G_\varepsilon \rangle$ and do the following. First replace the shape predicates in all judgements by

[4] In the sense the the judgement in question is the conclusion of one inference step and among the premises for the other.

$\langle Y' \mid G' \rangle$. Then remove all Eps rules (which now have identical judgements as premise and conclusion). The remaining rules are in the rewritten fragment are still valid; in the case of Pfx this is because the only proper edges in $G_\varepsilon$ that can be used in the fragment are labelled $C_i$ for some $i$, and we have just derived that $G'$ contains $Y' \xrightarrow{C_i} Y'$ for all $C_i$. Now attach the derivations of $\vdash R_i : \langle Y' \mid G' \rangle$ to the top of the rewritten prefix, and we get a full derivation of $\vdash P : \langle Y' \mid G' \rangle$.

2 $\pi = a$ and $\mathcal{T}(a) = \texttt{<}C_1.\cdots.C_k\texttt{>}$. As before, we have $\mathcal{S}\,P^* = \mathcal{S}(a).(\cdots \mid \mathcal{S}\,R_i^* \mid \cdots)$. If $\vdash \mathcal{S}\,P^* : \langle Y' \mid G' \rangle$ for some $Y'$, then $G'$ must contain a chain

$$Y' = Y'_0 \xrightarrow{C_1} Y'_1 \xrightarrow{C_2} \cdots \xrightarrow{C_k} Y'_k$$

We have $\vdash \mathcal{S}\,R_i^* : \langle Y'_k \mid G' \rangle$ and therefore $\vdash R_i : \langle Y'_k \mid G' \rangle$. The derivation fragment itself can be converted to a derivation for $\vdash P : \langle Y' \mid G' \rangle$ by removing the Eps rules at its top and bottom and replacing each $\langle Z_i \mid G_\varepsilon \rangle$ with $\langle Y'_i \mid G' \rangle$.

3. Trivial: $\pi = a$ and $P = b.R_1$ where $\mathcal{S}(a) = b$.

4. Trivial: $\pi = C$ and $P = C'.R_1$ where $\mathcal{T}C = \texttt{<}C'\texttt{>}$ and $C' = \mathcal{S}\,C$.

5. Trivial: $\mathcal{T}\pi = \pi = (\vec{a}) = p = \mathcal{S}p$ and $P = p.R_1$.

6. $\pi = \langle \mu_1, \ldots, \mu_k \rangle$ and $P = \langle M_1, \ldots, M_k \rangle.R_1$ such that $\vdash M_i : \mathcal{T}\mu_i$. We have $P^* = \langle \lceil \mu_1 \rceil, \ldots, \lceil \mu_k \rceil \rangle.R_1^*$, so $\mathcal{S}\,P^* = \langle \mathcal{S}\lceil \mu_1 \rceil, \ldots, \mathcal{S}\lceil \mu_k \rceil \rangle.(\mathcal{S}R_1^*)$. If $\vdash \mathcal{S}P^* : \langle Y' \mid G' \rangle$, then $G'$ must contain an edge $Y' \xrightarrow{\langle \mu'_1, \ldots, \mu'_k \rangle} Y''$ such that $\vdash \mathcal{S}R_1^* : \langle Y'' \mid G' \rangle$ and therefore $\vdash R_1 : \langle Y'' \mid G' \rangle$. Furthermore for each $i$, $\vdash \mathcal{S}\lceil \mu_i \rceil : \mu'_i$, so by Lemma A.1, $\mathcal{T}\mu_i \leq \mu'_i$. Therefore $\vdash M_i : \mu'_i$, and so $\vdash P : \langle Y' \mid G' \rangle$, as required. □

Now we return to the proof of Theorem 4.11. To prove $\mathcal{T}\sigma \leq \sigma'$, let $P \in [\![\mathcal{T}\sigma]\!]$ be given. We then must establish that $P \in [\![\sigma']\!]$.

By Lemma 4.6 we have $\vdash P : \langle X \mid G_\varepsilon \rangle$, so by Lemma A.2 there is a $P^* \in [\![\sigma]\!]$ such that $\vdash \mathcal{S}P^* : \sigma'$ implies $\vdash P : \sigma'$. But by assumption, $P^* \in [\![\sigma]\!]$ implies $\vdash \mathcal{S}P^* : \sigma'$, so we have $\vdash P : \sigma'$, as required.

# B Proof of Theorem 5.7

The two parts of the theorem are Propositions B.1 and B.2 below.

**Proposition B.1.** *If $\sigma$ is syntactically closed, then it is semantically closed.*

Proof. Assume $\vdash P_0 : \sigma$ and prove by induction on the derivation of $P_0 \hookrightarrow P_1$ that $\vdash P_1 : \sigma$:

- For the axiom $a[\text{in } b.P \mid Q] \mid b[R] \hookrightarrow b[a[P \mid Q] \mid R]$, the derivation of $\vdash P_0 : \sigma$ must end in

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash P : \langle Y \mid G\rangle}{\vdash \text{in } b.P : \langle X \mid G\rangle} \qquad \vdash Q : \langle X \mid G\rangle}{\vdash \text{in } b.P \mid Q : \langle X \mid G\rangle}}{\vdash a[\text{in } b.P \mid Q] : \langle X_0 \mid G\rangle} \qquad \dfrac{\vdash R : \langle Z \mid G\rangle}{\vdash b[R] : \langle X_0 \mid G\rangle}}{\vdash P_0 : \sigma = \langle X_0 \mid G\rangle}$$

  so the premises of point 1 of Definition 5.2 are satisfied. Therefore there is $Z \xrightarrow{\text{amb } a} X' \in G$ such that $\langle X \mid G\rangle \leq \langle X' \mid G\rangle$ (and therefore $\vdash Q : \langle X' \mid G\rangle$) and $\langle Y \mid G\rangle \leq \langle X' \mid G\rangle$ (and therefore $\vdash P : \langle X' \mid G\rangle$).

  We can now construct the derivation

$$\dfrac{\dfrac{\dfrac{\vdash P : \langle X' \mid G\rangle \qquad \vdash Q : \langle X' \mid G\rangle}{\vdash P \mid Q : \langle X' \mid G\rangle}}{\vdash a[P \mid Q] : \langle Z \mid G\rangle} \qquad \vdash R : \langle Z \mid G\rangle}{\vdash a[P \mid Q] \mid R : \langle Z \mid G\rangle}$$
$$\vdash P_1 : \langle X_0 \mid G\rangle$$

- The cases for the out and open axioms are similar and omitted.

- For the axiom $\langle M_1, \ldots, M_n\rangle.P \mid (a_1, \ldots, a_n).Q \hookrightarrow P \mid [a_i \mapsto M_i]_{1\leq i\leq n}Q$, the derivation of $\vdash P_0 : \sigma$ must end in

$$\dfrac{\dfrac{\vdash P : \langle Y \mid G\rangle}{\vdash \langle M_1, \ldots, M_n\rangle.P : \langle X_0 \mid G\rangle} \qquad \dfrac{\vdash Q : \langle Z \mid G\rangle}{\vdash (a_1, \ldots, a_n).Q : \langle X_0 \mid G\rangle}}{\vdash P_0 : \sigma = \langle X_0 \mid G\rangle}$$

  where $G$ contains $X_0 \xrightarrow{\langle \mu_1, \ldots, \mu_n\rangle} Y$ with $\vdash M_i : \mu_i$ for each $i$. The conditions in point 4 of Definition 5.2 are met, and we get $\langle Y \mid G\rangle \leq \langle X_0 \mid G\rangle$ (hence $\vdash P : \langle X_0 \mid G\rangle$) and $[a_i \mapsto \mu_i]_{1\leq i\leq n}\langle Z \mid G\rangle \leq \langle X_0 \mid G\rangle$ and hence, by Theorem 4.8, $\vdash [a_i \mapsto M_i]_{1\leq i\leq n}Q : \langle X_0 \mid G\rangle$).

  We can now construct the derivation

$$\dfrac{\vdash P : \langle X_0 \mid G\rangle \qquad \vdash [a_i \mapsto M_i]_{1\leq i\leq n}Q : \langle X_0 \mid G\rangle}{\vdash P_1 : \langle X_0 \mid G\rangle}$$

- For the rule $P \hookrightarrow Q \Longrightarrow a[P] \hookrightarrow a[Q]$, the derivation of $\vdash P_0 : \sigma$ must end in

$$\dfrac{\vdash P : \langle Y \mid G\rangle}{\vdash P_0 : \sigma = \langle X \mid G\rangle}\text{Pfx}$$

  where $G$ contains $X \xrightarrow{\text{amb } a} Y$. Therefore $\text{active}(\langle Y \mid G\rangle) \subseteq \text{active}(\sigma)$ and so $\langle Y \mid G\rangle$ is syntactically closed. The induction hypothesis now gives us $\vdash Q : \langle Y \mid G\rangle$ which leads immediately to $\vdash P_1 : \sigma$.

- The case for $P \hookrightarrow Q \Longrightarrow P \mid R \hookrightarrow Q \mid R$ is similar but simpler.
- For the rule $P \equiv P' \wedge P' \hookrightarrow Q' \wedge Q' \equiv Q \Longrightarrow P \hookrightarrow Q$, Theorem 3.2 gives us $\vdash P' : \sigma$; the induction hypothesis gives $\vdash Q' : \sigma$; and yet another application of Theorem 3.2 gives us $\vdash P_2 = Q : \sigma$. □

**Proposition B.2.** *If $\sigma$ is semantically closed, discrete, and trim, then $\sigma$ is syntactically closed.*

Proof. Assume that $\sigma = \langle X \mid G \rangle$ is *not* syntactically closed. We must then find $P$ and $Q$ such that $P \hookrightarrow Q$ and $P \in [\![\sigma]\!]$ but $Q \notin [\![\sigma]\!]$

By assumption there is a $X_0$ that causes one of the rules in Definition 5.2 to break. Because $X_0 \in \mathrm{active}(\sigma)$, $G$ contains a chain of nodes

$$X = X_k \xrightarrow{\text{amb } a_{k-1}} X_{k-1} \xrightarrow{\text{amb } a_{k-2}} \cdots \xrightarrow{\text{amb } a_0} X_0$$

Proceed by induction on $k$. In the base case, divide according to which of the conditions of Definition 5.2 does not hold at $X_0$:

1. The idea is to set $P = a[\text{in } b.R_1 \mid R_2] \mid b[R_3]$ and $Q = b[a[R_1 \mid R_2] \mid R_3]$, choosing $R_1$, $R_2$, and $R_3$ large enough to prevent $Q$ from matching $\langle X_0 \mid G \rangle$.

   Using the notation of Definition 5.2(1), we know that $X$, $Y$, and $Z$ exist, but that there is no suitable $X'$.

   Define $\mathbf{X} = \{ X' \mid Z \xrightarrow{\text{amb } a} X' \}$, and choose $R_1$ and $R_2$ such that $R_1 \mid R_2$ cannot match any $X' \in \mathbf{X}$. This is possible because every $X' \in \mathbf{X}$ must fail one of the conditions $\langle X \mid G \rangle \leq \langle X' \mid G \rangle$ or $\langle Y \mid G \rangle \leq \langle X' \mid G \rangle$. Thus we can choose a $R'_{X'}$ that is either in $[\![\langle X \mid G \rangle]\!]$ or $[\![\langle Y \mid G \rangle]\!]$ but not in $[\![\langle X' \mid G \rangle]\!]$. Let $R_1$ be the parallel composition of all of the $R'_{X'}$'s that match $Y$ and $R_2$ be the parallel composition of those that match $X$.

   Define $\mathbf{Z} = \{ Z' \neq Z \mid X_0 \xrightarrow{\text{amb } b} Z' \}$, and choose $R_3$ such that it cannot match any $Z' \in \mathbf{Z}$. This is possible because $G$ is assumed to be trim, so for each $Z' \in \mathbf{Z}$ it holds that $\langle Z' \mid G \rangle \not\leq \langle Z \mid G \rangle$. We can therefore choose a $R'_{Z'}$ that is in $[\![\langle Z \mid G \rangle]\!]$ but not in $[\![\langle Z' \mid G \rangle]\!]$. Let $R_3$ be the parallel composition of all of these $R'_{Z'}$ terms.

   Now it is clear that $\vdash P : \langle X_0 \mid G \rangle$, but due to the construction of $R_1$, $R_2$, and $R_3$ there can be no derivation of $\vdash Q : \langle X_0 \mid G \rangle$.

2. The idea is to set $P = a[b[\text{out } a.R_1 \mid R_2]]$ and $Q = a[0] \mid b[R_1 \mid R_2]$, choosing $R_1$ and $R_2$ large enough to prevent $Q$ from matching $\langle X_0 \mid G \rangle$.

   Using the notation of Definition 5.2(2), we know that $X$, $Y$, and $Z$ exist, but that there is no suitable $Y'$.

   Define $\mathbf{Y} = \{ Y' \mid X_0 \xrightarrow{\text{amb } b} Y' \}$, and choose $R_1$ and $R_2$ such that $R_1 \mid R_2$ cannot match any $Y' \in \mathbf{Y}$. This is analogous to the choice of $R_1$ and $R_2$ in the previous case.

   Now it is clear that $\vdash P : \langle X_0 \mid G \rangle$, but due to the construction of $R_1$ and $R_2$, $\nvdash Q : \langle X_0 \mid G \rangle$.

3. Set $P = \mathsf{open}\ a.R_1 \mid a[R_2]$ and $Q = R_1 \mid R_2$, where $R_1$ and $R_2$ are to be chosen large enough. The choice of $R_1$ and $R_2$ is similar to one step of the choice of $R_1$ and $R_2$ in the previous two cases.

4. We want to set $P = \langle M_1, \ldots, M_k\rangle.R_1 \mid (a_1, \ldots, a_k).R_2$ and $Q = R_1 \mid [a_i \mapsto M_i]_{1\le i\le k} R_2$, where $R_1$, $R_2$ and the $M_i$'s are to be chosen "large" enough. If Definition 5.2(4) fails at the condition $\langle Y \mid G\rangle \le \langle X_0 \mid G\rangle$, choose $R_1 \in [\![\langle Y \mid G\rangle]\!] \setminus [\![\langle X_0 \mid G\rangle]\!]$ and let $R_2 = 0$, $M_i = \varepsilon$.

   Otherwise, just let $R_1 = 0$. Set $\mathcal{T} = [a_i \mapsto \mu_i]_{1\le i\le k}$. Because G is assumed to be discrete, $\mathcal{T}$ is also discrete. We know that $\mathcal{T}\langle Z \mid G\rangle \not\le \langle X_0 \mid G\rangle$. By Theorem 4.11 we then get $\mathcal{S}$ and $R_2$ such that $\vdash R_2 : \langle Z \mid G\rangle$ and $\vdash \mathcal{S} : \mathcal{T}$ but not $\vdash \mathcal{S}R_2 : \langle X_0 \mid G\rangle$. Set $M_i = \mathcal{S}a_i$ for all $i$, and we are through: $\vdash P : \langle X_0 \mid G\rangle$ but not $\vdash Q : \langle X_0 \mid G\rangle$.

Now for the inductive case. Because G is trim, for each $X_k \xrightarrow{\mathsf{amb}\ a_{k-1}} Y \in G$ such that $Y \neq X_{k-1}$ we know that $\langle X_{k-1} \mid G\rangle \not\le \langle Y \mid G\rangle$. Choose therefore $R_Y \in [\![\langle X_{k-1} \mid G\rangle]\!] \setminus [\![\langle Y \mid G\rangle]\!]$ for each such $Y$, and let $R$ be the parallel composition of all these $R_Y$'s. Then, for all $R'$, $\vdash a_{k-1}[R' \mid R] : \langle X_k \mid G\rangle$ if and only if $\vdash R' : \langle X_{k-1} \mid G\rangle$.

The induction hypothesis gives $P' \hookrightarrow Q'$ such that $P' \in [\![\langle X_{k-1} \mid G\rangle]\!]$ but $Q' \notin [\![\langle X_{k-1} \mid G\rangle]\!]$. Now let $P = a_{k-1}[P' \mid R]$ and $Q = a_{k-1}[Q' \mid R]$. □

# C Proofs about infinite shape graphs

**Lemma C.1.** *Let* G *be finitary and modest. There exists a number* $k$ *such that no chain in* G *that does not repeat any node name can be longer than* $k$ *edges.*

Proof. Use induction on the maximal $\mathbf{S}(\pi)$ for any $\pi$ that appears in the chain.

Consider repetition-free chains whose maximal $\mathbf{S}(\pi)$ value is $x$. Since G is finitary, there are only finitely many possible $\pi$ with $\mathbf{S}(\pi) = x$. Consider each of them in turn, and look at the way G is modest for $\pi$. If it satisfies finite depth, then there cannot be more than $n_\pi$ copies of $\pi$ in any chain. If it satisfies monomorphic restriction, set $n_\pi = 1$ — a chain with no node repetition can contain at most one $\pi$. Set $m = \sum_{\mathbf{S}(\pi)=x} n_\pi$; this is an upper bound on the number of prefix types with $\mathbf{S}(\pi) = x$ in the chain.

The induction hypothesis gives us an upper bound $k'$ of the length of a (repetition-free) subchain composed entirely of $\pi$'s with $\mathbf{S}(\pi) < x$. Therefore, a chain with $\mathbf{S}(\pi) \le x$ can have length at most $(m+1)k' + m$, which is finite, as claimed. □

**Proof of Proposition 5.15.** Let $d$ be the number of different prefix types in G, and $k$ be the bound on the length of a repetition-free chain in G derived in Lemma C.1.

Let an arbitrary X be given. Because $k$ is finite, $[\![\langle X \mid G\rangle]\!]$ can also be realized as $[\![\langle X \mid G'\rangle]\!]$, where $G'$ is consists of a tree rooted at X plus some direct back edges.

The tree in $G'$ has finite *height* but may still be infinitely *branching*. We now argue that any infinite branching can be pruned away without changing the meaning of $\langle X \mid G \rangle$. We start at the leaves of the tree and work up towards the root by induction. The invariant is that when a all the set of subtrees with height less than $i$ has been processed, they fall into a most $L_i$ isomorphism classes (considering two back edges to be isomorphic when they span the same number of levels and are decorated with the same prefix type), where $L_0 = 0$ and $L_i = 2^{(k+L_{i-1})d}$.

At level $i$, each node $Y$ can have $(k+L_i)d$ different "outgoing items", namely for each possible $\pi$ up to $k$ different back edges and a forward edge for each of the $L_i$ isomorphism classes at the previous level. If some isomorphism class is represented more than once, we can safely remove all but one of the representatives without changing the meaning of $\langle X \mid G' \rangle$.

When we reach the root at level $k$, we have found that $\langle X \mid G' \rangle$ has the same meaning as one of the $L_{k+1}$ possible isomorphism classes at this level. Since $L_{k+1}$ is finite (though rather large), this proves the proposition. □

**Lemma C.2.** *Proposition 3.11 is true for finitary modest graphs as well as for finite (but not necessarily modest) ones.*

Proof. The proof of Proposition 3.11 depends on $\mathbf{Y}$ being finite in the argument that the iteration that produces $\mathbf{Y}_k$ will eventually stop. However, this argument can be replaced by one based on the number of different *meanings* of elements of $\mathbf{Y}_i$, which is also strictly decreasing and, thanks to Proposition 5.15, is always finite.

All other parts of the proof work directly with finitary graphs as well as finite ones. □

**Lemma C.3.** *For any message type $\mu$, the set $\{ \mu' \mid \mu' \leq \mu \}$ is finite.*

**Lemma C.4.** *Let $\mathbf{m}$ be a nonempty, possibly infinite set of message types, and let $\mathbf{M} = \bigcap_{\mu \in \mathbf{m}} [\![\mu]\!]$. If $\mathbf{M}$ is nonempty, then there is a $\mu'$ such that $[\![\mu']\!] = \mathbf{M}$.*

Proof. Assume that $\mathbf{M}$ is nonempty. Then we can choose an $M_0 \in \mathbf{M}$ such that $\vdash M_0 : \mu$ for each $\mu$ in $\mathbf{m}$.

If $\mathbf{m}$ contains a $\mu_0 = \{a\}$, then $\vdash M_0 : \mu_0$ implies $M_0 = a$. Since $M_0 \in \mathbf{M} \subseteq [\![\mu_0]\!] = \{M_0\}$, we have $\mathbf{M} = [\![\mu_0]\!]$, and we are done.

If $\mathbf{m}$ contains a $\mu_0 = <\vec{C}>$, then $M_0 \notin \boxed{a}$, $M_0 \equiv \vec{C}.0$. But by inspection of the rules for $\vdash M : \mu$ this means that $[\![\mu_0]\!] \subseteq [\![\mu]\!]$ whenever $\vdash M_0 : \mu$. Therefore $\mathbf{M} = [\![\mu_0]\!]$, and we are done.

Otherwise all $\mu \in \mathbf{m}$ have the form $\{\vec{C}\}*$. Then set $\mu' = \{C_1, \ldots, C_k\}*$, where $C_1$ to $C_k$ are those capabilities that appear in *all* $\mu \in \mathbf{m}$. □

**Lemma C.5.** *Let $(\pi_i)_{i \in I}$ be a (finite or infinite but nonempty) family of prefix types. If $\bigcap_{i \in I} [\![\pi_i]\!]$ is nonempty, there is a $\pi$ such that $[\![\pi]\!] = \bigcap_{i \in I} [\![\pi_i]\!]$.*

Proof. This follows easily from Lemma C.4. □

**Proof of Proposition 5.16.** Let $\sigma_i = \langle X_i \mid G_i \rangle$. Without loss of generality we can assume that all node names used in any $\sigma_i$ are contained in a common, countable, set **X**.

We will construct an infinite graph $G_\infty$ whose node names are all of the families $(X_i)_{i\in I}$. Let **G** be $\prod_{i\in I} G_i$ – that is, the set of all families of edges $(e_i)_{i\in I}$, such that $e_i \in G_i$. The infinite graph $G_\infty$ will contain zero or one edge for each $(Y_i \xrightarrow{\pi_i} Z_i)_{i\in I} \in \mathbf{G}$: Apply Lemma C.5 to $(\pi_i)_{i\in I}$. If that results in a $\pi$, then $G_\infty$ must contain the edge $(Y_i)_{i\in I} \xrightarrow{\pi} (Z_i)_{i\in I}$.

This completes the description of $G_\infty$; now set $\sigma = \langle (X_i)_{i\in I} \mid G_\infty \rangle$. We must prove that this satisfies the condition in the statement of the proposition:

- $[\![\sigma]\!] \subseteq \bigcap_{i\in I} [\![\sigma_i]\!]$: Assume $\vdash P : \sigma$ and let $k \in I$ be given. The derivation of $\vdash P : \sigma$ can be projected to one for $\vdash P : \sigma_k$ by replacing each shape predicate $\langle (Y_i)_{i\in I} \mid G_\infty \rangle$ in it by $\langle Y_k \mid G_k \rangle$. If must be checked that the projected derivation is still valid, but this is easy: The only rule that might create trouble is Pfx, and by construction, whenever $G_\infty$ contains $(Y_i)_{i\in I} \xrightarrow{\pi} (Z_i)_{i\in I}$ and $\vdash p : \pi$, there will be a $\pi_k$ such that $Y_k \xrightarrow{\pi_k} Z_k$ and $\vdash p : \pi_k$.

- $[\![\sigma]\!] \supseteq \bigcap_{i\in I} [\![\sigma_i]\!]$: Assume $\vdash P : \sigma_i$ for each $i$. All of these judgements have derivations whose shape follows the syntax of $P$. By construction of $G_\infty$ they can be combined point-wise into a derivation of $\vdash P : \sigma$.

- $G_\infty$ is finitary: Since $I$ is assumed to be non-empty, select and fix some index $i_0 \in I$. Each $\pi$ found in $G_\infty$ corresponds to a $\pi'$ in $G_{i_0}$ such that $\pi \leq \pi'$. Lemma C.3 imply that for each $\pi'$ there are only finitely many possible $\pi$'s. Because $G_{i_0}$ is finite and hence finitary, this proves that $G_\infty$ is finitary.

- $G_\infty$ is modest: Let $\pi$ be given. If at least one $G_i$ satisfies the "finite depth" criterion for $\pi$ with limit $n$, then $G_\infty$ also does, because any chain in $G_\infty$ that violated the criterion would project to one in $G_i$ that also did.

  If, on the other hand, all $G_i$ satisfy the "monomorphic recursion" criterion, then $G_\infty$ also does. Namely, any chain in $G_\infty$ with identically classified $\pi$'s at each end projects to a similar chain in each $G_i$, which proves the pointwise identity of the chain's $X_1$ and $X_k$ nodes.

- $G_\infty$ is discrete: Again this follows because each chain of identical capabilities in $G_\infty$ projects to a similar chain in each $G_i$, which are supposed to be discrete themselves. □

**Lemma C.6.** *Let $G_\infty$ be a finitary and modest shape graph and let $G'$ be finite and trim. Assume that $[\![\langle X \mid G_\infty \rangle]\!] = [\![\langle X' \mid G' \rangle]\!]$ and $X' \xrightarrow{\pi'} Y' \in G'$. Then there exists $X \xrightarrow{\pi} Y \in G_\infty$ such that $\pi =_{(\leq)} \pi'$ and $[\![\langle Y \mid G_\infty \rangle]\!] = [\![\langle Y' \mid G' \rangle]\!]$.*

Proof. Because $\langle X' \mid G' \rangle \leq \langle X \mid G_\infty \rangle \leq \langle X' \mid G' \rangle$, Lemma C.2 applied twice gives us

1. $X \xrightarrow{\pi} Y \in G_\infty$, with $\pi' \leq \pi$ and $\langle Y' \mid G' \rangle \leq \langle Y \mid G_\infty \rangle$.

2. $X' \xrightarrow{\pi''} Y'' \in G'$, with $\pi \leq \pi''$ and $\langle Y \mid G_\infty \rangle \leq \langle Y'' \mid G' \rangle$.

In total, $\langle Y' \mid G' \rangle \leq \langle Y \mid G_\infty \rangle \leq \langle Y'' \mid G' \rangle$ and $\pi' \leq \pi''$. But because $G'$ is trim, this implies that $Y' = Y''$, so $Y$ does have the desired property. □

**Proof of Proposition 5.17.** Let $\sigma_\infty = \langle Z \mid G_\infty \rangle$. By Proposition 5.15, $\sim_{G_\infty}$ divides the set of node names into finitely many equivalence classes. Select a representative element in each class, and let $\overline{X}$ stand for the representative for $X$'s class.

Let $G'_0 = \{ \overline{X} \xrightarrow{\pi} \overline{Y} \mid X \xrightarrow{\pi} Y \in G_\infty \}$. It is clear by Proposition 5.15 that $G'_0$ is finite. It is also easy to prove by induction on term structure that $[\![\langle \overline{X} \mid G'_0 \rangle]\!] = [\![\langle X \mid G_\infty \rangle]\!]$.

Now let $G'$ be a trim equivalent of $G'_0$ (by Lemma 5.6), and set $\sigma' = \langle \overline{Z} \mid G' \rangle$. Then $[\![\sigma']\!] = [\![\sigma_\infty]\!]$.

It remains to prove that $G'$ is discrete and modest. That is true because every chain in $G'$ corresponds to a chain with $=_{(\leq)}$-equivalent prefix types in $G_\infty$, which can be seen by using Lemma C.6 for each link in the chain, starting from the beginning.

(For discreteness, a repetition of a capability $C$ in the $G'$ chain will map to a repetition of the same capability $C$ in the $G_\infty$ chain, because $=_{(\leq)}$ relates a capability only to itself.) □

# D Proof of Theorem 6.3

**Definition D.1.** *Define the relation* $B \vdash P : \sigma$ *to hold if no* $b \in B$ *is bound by* $(\vec{a})$ *within* $P$*, and there is a* $P'$ *such that* $B \vdash P \curvearrowright P'$ *and* $\vdash P' : \sigma$.

Clearly $P \in [\![\sigma/B]\!]$ implies $B \vdash P : \sigma$.

**Proof of Theorem 6.3(1).** It is easy to see that $P \equiv Q$ implies that the free variables in $P$ and $Q$ are the same. That $[\![\sigma/B]\!]$ is closed under $\equiv$ now follows from the following lemma. □

**Lemma D.2.** *If* $P \equiv Q$*, then* $B \vdash P : \sigma \iff B \vdash Q : \sigma$.

Proof. By induction on the *height* of the derivation of $P \equiv Q$. The case for $\frac{P \equiv Q}{p.P \equiv p.Q}$ is typical. Since this rule is clearly symmetric, we only show the "⇒" direction.

Assume $B \vdash p.P : \sigma$. Then there is $P''$ such that $B \vdash p.P \curvearrowright P''$ and $\vdash P'' : \sigma$. By inspection of the $\curvearrowright$ rules, $P''$ must be $p.P'$ for some $P'$ with $B \vdash P \curvearrowright P'$, and then $\vdash P'' = p.P' : \sigma$ must be proved by rule Pfx, with premise $\vdash P' : \sigma'$. Now $B \vdash P : \sigma'$, and by the induction hypothesis $B \vdash Q : \sigma'$. That is, for some $Q'$, $B \vdash Q \curvearrowright Q'$ and $\vdash Q' : \sigma'$. But then $B \vdash p.Q \curvearrowright p.Q'$ and $\vdash p.Q' : \sigma$. Thus $B \vdash p.Q : \sigma$, as required.

Most of the cases are similar, and omitted.

For the rule $\frac{P \equiv Q}{\nu a.P \equiv \nu a.Q}$, assume $B \vdash \nu a.P : \sigma$. Then there is $b \in B$ such that $B \vdash [a \mapsto b]P : \sigma$. By a renaming $a$ to $b$ everywhere in the derivation of $P \equiv Q$ we get a derivation of $[a \mapsto b]P \equiv [a \mapsto b]Q$ with the same height. Therefore

we can use the induction hypothesis to get $B \vdash [a \mapsto b]Q : \sigma$. This implies $B \vdash \nu a.Q : \sigma$, as required.

In the case for $P \mid \nu a.Q \equiv \nu a(P \mid Q)$, the essential step is noticing that $[a \mapsto b]P = P$ because $a$ is not free in $P$. Similarly for $a[\nu b.P] \equiv \nu b.a[P]$.

For $\nu a.0 \equiv 0$, the $\Rightarrow$ direction is trivial, and the $\Leftarrow$ direction goes through because a $B$ is in general assumed to be nonempty. □

**Proof of Theorem 6.3(2).** By the following lemma. □

**Lemma D.3.** *If $\sigma \leq \sigma'$, then $B \vdash P : \sigma$ implies $B \vdash P : \sigma'$.*

Proof. This follows from the obvious fact that the function

$$\mathbf{P} \mapsto \{P \mid \text{no } b \in B \text{ is bound by } (\vec{a}) \text{ in } P \ \wedge \ \exists P' \in \mathbf{P} : B \vdash P \curvearrowright P'\}$$

is monotonic. □

**Proof of Theorem 6.3(3).** By the following lemma. □

**Lemma D.4.** *If $\sigma$ is syntactically closed, then $B \vdash P_0 : \sigma$ and $P_0 \hookrightarrow P_1$ imply $B \vdash P_1 : \sigma$.*

Proof. By induction on a lexicographic order with the *height* of the derivation of $P_0 \hookrightarrow P_1$ in the most significant position and the size of $P_0$ in the least significant one.

Most cases are completely similar to the corresponding ones from Proposition B.1. The part of the typing derivation for $P_0$ that is unraveled in each case does not contain any $\nu$'s, and therefore the corresponding unraveling of the derivation for $B \vdash P_0 : \sigma$ is trivial too. Use Lemma D.3 wherever the proof of Proposition B.1 appeals to the definition of $\leq$.

In the case for the communication axiom, we need the property that $B \vdash Q \curvearrowright Q'$ implies $B \vdash [a_i \mapsto M_i]_{1 \leq i \leq n} Q \curvearrowright [a_i \mapsto M_i]_{1 \leq i \leq n} Q'$. That is not true in general, but because of the assumption that no $b \in B$ appear in a $(\vec{a})$ action, none of the $a_i$'s will be in $B$, and the property thus holds anyway.

In the case for $P \equiv P' \hookrightarrow Q' \equiv Q$, use Lemma D.2 instead of Theorem 3.2.

We need to add a new case for the rule $\frac{P \hookrightarrow Q}{\nu a.P \hookrightarrow \nu a.Q}$. From the assumption that $B \vdash \nu a.P : \sigma$ we get a $b \in B$ such that $B \vdash [a \mapsto b]P : \sigma$.

We can now prove that $P \hookrightarrow Q$ implies $[a \mapsto b]P \hookrightarrow [a \mapsto b]Q$, with a derivation of the same height. This is easily proved by induction on the derivation of $P \hookrightarrow Q$. The only tricky step in this sub-induction is the one for the communication axiom, where the substitution would collapse the $(\vec{a})$ action to $\bullet$ if the action contained $a$. Fortunately, we have stipulated in general that $P$, which is the body of a $\nu a$ abstraction, does not contain an $a$ in this position.

Since $[a \mapsto b]P$ is strictly smaller than $\nu a.P$, we can now use the induction hypothesis to get $B \vdash [a \mapsto b]Q : \sigma$. From there we easily reach $B \vdash \nu a.Q : \sigma$, and we are done. □

**Proposition D.5.** *Let $P \in [\![\sigma/B]\!]$ and assume that $B' \vdash P \curvearrowright P'$ can be derived such that each $b \in B'$ is used exactly once (and is not free in $P$). There is a $\sigma'$ such that $\vdash P' : \sigma'$ and $[\![\sigma'/B']\!] \subseteq [\![\sigma/B]\!]$. Furthermore, if $\sigma$ is syntactically closed, discrete, and/or modest, then $\sigma'$ is too.*

Proof. $P \in [\![\sigma/B]\!]$ means that there is a $Q$ such that $B \vdash P \curvearrowright Q$ and $\vdash Q : \sigma$. Without loss of generality we can assume that $B \supseteq B'$; if not, then adding the missing ones to $B$ will only make it harder for a term to match $\sigma/B$ and therefore strengthen our result. Call names in $B \setminus B'$ **forbidden** names.

The derivations of $B \vdash P \curvearrowright Q$ and $B' \vdash P \curvearrowright P'$ have the same shape, dictated by $P$'s syntax; they differ only in which $b$'s they chose at each of the $\nu$'s in $P$. In this way each $b' \in B'$ corresponds to exactly one $b \in B$; let $\mathcal{S}$ be $[\, b' \mapsto b \mid b' \text{ corresponds to } b \,]$.

In other words, $\mathcal{S}$ is the unique term substitution with domain $B'$ such that $Q = \mathcal{S}P'$.

Now define the function $\psi$ from names, capabilities, message types, prefix types and edges to sets of names, capabilities, etc.:

$$\begin{aligned}
\psi(a) &= \begin{cases} \{\, b' \in B' \mid a = \mathcal{S}(b') \,\} & \text{when } a \in B \\ \{a\} & \text{otherwise} \end{cases} \\
\psi(\bullet) &= \{\bullet\} \\
\psi(O\ a) &= \{\, O\ a' \mid a' \in \psi(a) \,\} \\
\psi(\{C_1, \ldots, C_k\}*) &= \{\{\psi(C_1) \cup \cdots \cup \psi(C_k)\}* \text{ with duplicates removed}\} \\
\psi(<C_1.\cdots.C_k>) &= \{\, <C_1'.\cdots.C_k'> \mid C_i' \in \psi(C_i) \,\} \\
\psi(\{a\}) &= \{\, \{a'\} \mid a' \in \psi(a) \,\} \\
\psi(\langle \mu_1, \ldots, \mu_n \rangle) &= \{\, <\mu_1'.\cdots.\mu_n'> \mid \mu_i' \in \psi(\mu_i) \,\} \\
\psi((a_1, \ldots, a_k)) &= \begin{cases} \varnothing & \text{if some } a_i \in B \\ \{(a_1, \ldots, a_k)\} & \text{otherwise} \end{cases} \\
\psi(X \xrightarrow{\pi} Y) &= \{\, X \xrightarrow{\pi'} Y \mid \pi' \in \psi(\pi) \,\}
\end{aligned}$$

and define

$$\begin{aligned}
G' &= \bigcup_{e \in G} \psi(e) \quad \text{where } \sigma = \langle X \mid G \rangle \\
\sigma' &= \langle X \mid G' \rangle
\end{aligned}$$

Note that $\psi(b)$ will be the empty set if $b \in B$ does not occur in $Q$. This emptiness will propagate to $\psi$ of larger objects that contain such a $b$, except if protected by $\{\cdots\}*$. It is easy to see that the result of $\psi$ never contains forbidden names, and that

1. $\mathcal{S}(a') = a \iff a' \in \psi(a)$ unless $a'$ is forbidden.
2. $\mathcal{S}\,C' = C \iff C' \in \psi(C)$ unless $C'$ contains forbidden names.
3. $\vdash \mathcal{S}\,M' : \mu \iff \exists \mu' \in \psi(\mu) : \vdash M' : \mu'$ unless $M'$ contains forbidden names.
4. $\vdash \mathcal{S}\,p' : \pi \iff \exists \pi' \in \psi(\pi) : \vdash p' : \pi'$ unless $p'$ contains forbidden names.
5. $\vdash \mathcal{S}\,R' : \sigma \iff \vdash R' : \sigma'$ unless $R'$ contains forbidden names.

In particular, because $P'$ contains no forbidden names and $\mathcal{S}\,P' = Q$, it must hold that $\vdash P' : \sigma'$, which is the first part of what we need to prove.

Now we must prove that $[\![\sigma'/B']\!] \subseteq [\![\sigma/B]\!]$. Let an arbitrary $R \in [\![\sigma'/B']\!]$ be given. Then we know an $R'$ such that $B' \vdash R \curvearrowright R'$ and $\vdash R' : \sigma'$. By construction, $\sigma'$ mentions no forbidden names, so $R'$ cannot do either. By definition of $\curvearrowright$, we see that $R$ itself must be free of forbidden names (every name that is free in $R$ is still free in $R'$).

Since $R'$ contains no forbidden names, we get $\vdash \mathcal{S}\,R' : \sigma$. But it is also easy to derive $B \vdash R \curvearrowright \mathcal{S}\,R'$ – one just chooses $\mathcal{S}(b')$ for each $\nu$ in $R$ where $B \vdash R \curvearrowright R'$ chose $b'$. Therefore $R \in [\![\sigma/B]\!]$, as required.

It remains to be seen that syntactic closure, discreteness, and modesty are inherited from $G$ to $G'$. For syntactic closure this is a straightforward though tedious matter of checking that the translation from $G$ to $G'$ preserves shape simulations, type substitution results, and finally syntactic closure. We omit the details.

Discreteness and modesty follow from the property that a chain in $G'$ projects to a similar chain in $G$. It can readily be checked from the definition of $\psi$ that $\psi(\pi_1) \ni \pi_1' =_{(\leq)} \pi_2' \in \psi(\pi_2)$ implies $\pi_1 =_{(\leq)} \pi_2$. □

**Proof of Theorem 6.3(4).** Without loss of generality we can assume that $P$ contains at least one $\nu$ – otherwise replace it with $P \mid \nu.0$ and use Theorem 6.3(1).

By assumption $P$ has some discrete and modest $\nu$-aware type $\tau/B$. Chose $P'$ and $B'$ such that $B' \vdash P \curvearrowright P'$ with each $b \in B'$ used exactly once. Then Proposition D.5 tells us that $P'$ has at least one discrete and modest type in the $\nu$-free theory.

By Theorem 5.13, this means that $P'$ has a type $\tau_0$ that is principal (among modest discrete types) in the $\nu$-free theory. We shall prove that $\tau_0/B'$ is principal for the original $P$.

Consider namely an arbitrary $\nu$-aware discrete modest type for $P$, say $\tau''/B''$. Applying Proposition D.5 again, we get a discrete modest $\tau'$ such that $[\![\tau'/B']\!] \subseteq [\![\tau''/B'']\!]$. We also have $\vdash P' : \tau'$, and since $\tau_0$ was principal, this implies $\tau_0 \leq \tau'$ and thus, by Theorem 6.3(2), $[\![\tau_0/B']\!] \subseteq [\![\tau'/B']\!]$. Taken together we have $[\![\tau_0/B']\!] \subseteq [\![\tau''/B'']\!]$, and since the latter was arbitrary this proves that $\tau_0/B'$ is indeed principal. □

# E Further restrictions on types

Though the system of modest types is strictly weaker than the unlimited system according to Definition 5.8, it is still flexible enough that further restrictions are probably necessary if one wants to have a complete type inference algorithm.

In this appendix we list – without further comments – some possible restrictions that are compatible with our principality results in the sense that they will be preserved by the constructions in Propositions 5.16 and 5.15. The type inference algorithm of Makholm and Wells [2004] is based on variants of the first two ones.

- **Determinism.** Require of all shape graphs G that if $X \xrightarrow{\pi} Y$ and $X \xrightarrow{\pi'} Y'$ are both in G and $\pi =_{(\leq)} \pi'$, then $Y = Y'$.

  Variants: Use this condition only for certain equivalence classes under $=_{(\leq)}$, or for a larger relation than $=_{(\leq)}$.

- **Monomorphic recursion.** For some or all $=_{(\leq)}$ equivalence classes, forbid the "finite depth" option in the definition of modesty.

- **Strong closure.** Require all shape graphs to be locally closed at *every* node X rather than just the active ones.

- **Unsequenced messages.** Forbid message types of the form $<C_1.\cdots.C_k>$.

- **Unsequenced capabilities.** Require that every edge of the shape $X \xrightarrow{C} Y$ has $X = Y$ unless C is an ambient boundary.

  Variants: Use this condition for another set of prefix types than the capabilities, as long as the set is closed under $=_{(\leq)}$.

- **Severe modesty.** Or use a larger (i.e., less discriminating) relation than $=_{(\leq)}$ in the definition of modesty.

- **Different stratification.** Use another definition of $\mathbf{S}(\pi)$. Our proofs do not depend on the particular definition set forth in Definition 5.11, except that it should be compatible with the $=_{(\leq)}$ relation. (The image of **S** can be any totally ordered set – it does not even need to be well-founded).