Technical Report HW-MACS-TR-0067
School of Mathematical and Computer Sciences
Heriot Watt-University

The Expressiveness of Poly✶, a Generic Process Calculus Type System

Jan Jakubův
Heriot-Watt University

J. B. Wells
Heriot-Watt University

May 15, 2009

Abstract

Meta✶ is a generic process calculus that can be instantiated by supplying rewriting rules defining an operational semantics to make numerous process calculi such as the π-calculus, the system of Mobile Ambients, and many of their variants. Poly✶ is a generic type system that makes a sound type system with principal types and a type inference algorithm for any instantiation of Meta✶.

This paper evaluates the expressiveness of Poly✶ by comparing it with two quite dissimilar static analysis systems in the literature. One comparison is with a typed version of Mobile Ambients defined by Cardelli and Gordon; we name this system TMA. The other comparison is with a flow analysis for BioAmbients by Nielson, Nielson, Priami, and Rosa; we name this flow analysis SABA. We instantiate Meta✶ and Poly✶ to the two process calculi (TMA and BioAmbients) and compare the types provided by Poly✶ with the predicates provided by the previous static analysis systems (the typing judgments of TMA and the flow analysis results of SABA). We show that Poly✶ types can express essentially the same information as the previous static analysis systems and can also express more precise information.

To do the comparisons, we needed to alter how Meta✶ handles α-conversion in order to develop a new method for handling name restriction in Poly✶.

1 Introduction

Many syntactic constructions have similar semantics in many process calculi found in the literature. Examples include *parallel composition* ("|"), prefixing a process with an action (sometimes called a *capability*) ("."), and *name restriction* ("ν"). Process calculi differ mainly in the set of actions and their meanings. Meta✶ [MW05, MW04] collects constructors shared among process calculi and provides a general syntax in which various actions can be encoded. Meta✶ is instantiated with a rewriting rule set $\mathcal{R}$ that specifies the meanings of

actions. Meta✶ can be instantiated to many calculi including for example the π-calculus [MPW92], Mobile Ambients [CG98], numerous variations of these, and other systems.

Poly✶ provides type systems for Meta✶. Instantiating Meta✶ by a rewriting rule set $\mathcal{R}$ makes a process calculus $C_{\mathcal{R}}$, and instantiating Poly✶ by $\mathcal{R}$ makes a type system for $C_{\mathcal{R}}$. Poly✶ provides *shape predicates* which describe possible process term configurations. A shape predicate π is a type for $C_{\mathcal{R}}$ if π's meaning is guaranteed by a simple test to be closed under rewriting with $\mathcal{R}$. Every instance of Poly✶ has desirable properties such as subject reduction, the existence of principal typings [Wel02], and an already implemented type inference algorithm.

This paper addresses the natural question of how expressive Poly✶ types are when compared to the predicates of other static analysis systems. First, we compare Poly✶ with a typed version of Mobile Ambients (MA) designed by Cardelli and Gordon which we call TMA. Because TMA is one of the first type systems for MA and many other MA type systems are based on it, TMA can viewed as the seminal MA type system. We embed TMA typing judgments in Poly✶. Second, we use a very different style of system to show Poly✶'s generality: the static analysis system for BioAmbients (SABA) designed by Nielson et al. We show that Poly✶ principal typings contain the information provided by SABA predicates, although they can (and generally will be) more precise.

This paper also refines α-conversion in Meta✶ and uses this to handle name restriction in Poly✶ more simply than before. This was important for the TMA comparison and simplified the SABA comparison. This change also yields uniform handling of ν-bound and input-bound names in Meta✶.

Sec. 1.1 describes notations, Sec. 2 describes Meta✶ and Poly✶, Sec. 3 provides a comparison with TMA, Sec. 4 compares with SABA, and we conclude in Sec. 5. Proofs of main theorems are found in App. A.

1.1 Notations and Preliminaries

Nat is the set of natural numbers. Let i, j, k range over Nat. We shall use one of the following statements (in this case with the same meaning) to express similar claims:

$$i, j, k \in \mathsf{Nat} ::= 0 \mid 1 \mid 2 \mid \cdots$$
$$i, j, k \in \mathsf{Nat} = \{0, 1, 2, \ldots\}$$

$\mathcal{P}_{\text{fin}}(U)$ is the set of all finite subsets of a set U, "$\setminus$" denotes set subtraction, and "$\times$" Cartesian product. A function f is a pair set such that $(u, v) \in f$ and $(u, w) \in f$ implies $v = w$. Let $u \mapsto v$ be an alternate pair notation used in functions. Given the function f and the sets U and V we suppose the following

$$
\begin{array}{rlll}
a, b \in & \mathsf{BasicName} & ::= \mathsf{a} \mid \mathsf{b} \mid \cdots \mid \mathsf{in} \mid \mathsf{out} \mid \mathsf{open} \mid \cdots \mid \mathsf{[]} \mid \bullet \mid \cdots \\
x, y \in & \mathsf{Name} & ::= a^i \\
F \in & \mathsf{Form} & ::= x_0 \ldots x_k \\
M \in & \mathsf{Message} & ::= F \mid \mathsf{0} \mid M_0.M_1 \\
E \in & \mathsf{Element} & ::= x \mid (x_1, \ldots, x_k) \mid \langle M_1, \ldots, M_k \rangle \\
A \in & \mathsf{Action} & ::= E_0 \ldots E_k \\
P, Q, R \in & \mathsf{Process} & ::= \mathsf{0} \mid A.P \mid (P \mid Q) \mid \nu(x).P \mid !P
\end{array}
$$

Figure 1: Syntax of Meta✶ processes.

definitions:

$\mathsf{dom}(f) = \{u \colon (u \mapsto v) \in f\}$	function's domain
$\mathsf{rng}(f) = \{v \colon (u \mapsto v) \in f\}$	function's range
$f^{-1} = \{(v, u) \colon (u \mapsto v) \in f\}$	inverse function/relation
$f[u \mapsto v] = \{(u' \mapsto v') \in f \colon u \neq u'\} \cup \{u \mapsto v\}$	function extension/replacement
$U \to V = \{f \subseteq (U \times V) \mid f \text{ is a function}\}$	all functions from U to V
$U \xrightarrow{\text{fin}} V = \{f \in (U \to V) \mid f \text{ is finite}\}$	all finite functions from U to V

2 Metacalculus Meta✶ and Generic Type System Poly✶

This section describes Meta✶ and Poly✶. Sec. 2.1 describes Meta✶ process syntax. Sec. 2.2 describes instantiating Meta✶ to specific process calculi. Sec. 2.3 briefly describes Poly✶.

Meta✶ [MW05, MW04] has changed: names are now built from basic names and α-conversion now preserves the underlying basic name (Sec. 2.1). This makes a new simple Poly✶ name restriction rule (Sec. 2.3) do the right thing. The previous handling of name restriction [MW04, Sec. 5.3] was overcomplicated and also inadequate for the comparison with TMA in Sec. 3.

2.1 General Syntax of Processes

Syntax of processes. Meta✶ process syntax supports simple embeddings of many calculi. Fig. 1 gives the syntax of Meta✶ entities. The metavariable Z ranges over all Meta✶ entities. A name a^i is a pair of a *basic name* a and a natural number i. When α-converting, we preserve the basic name and change the number. We write a instead of a^0 when no confusion can arise.

Processes are built from the null process "0" by prefixing with an action ("."), by parallel composition ("|"), by name restriction ("ν"), and by replication ("!"). Actions can support prefixes from various calculi such as π-calculus communication actions, MA capabilities, or MA ambient boundaries. Ambient boundaries are further supported by the notation "$x_1 \ldots x_i\,[P]\,y_1 \ldots y_j$" which abbreviates "$x_1 \ldots x_i\,[]\,y_1 \ldots y_j.P$" where "[]" is a (single) special name. For example, "$x\,[P]$" stands for "$x\,[].P$".

Process constructors have standard semantics. "0" is an inactive or finished process, "$A.P$" executes the action A and then continues as P, "$P \mid Q$" runs

Message decomposition operator:

$$(M_0.M_1)_* P = M_{0*}(M_{1*}P) \qquad \mathsf{0}_* P = P \qquad A_* P = A.P$$

Application of a substitution to names, forms, elements, and actions:

$$(E_0 \ldots E_k)\dot{\sigma} = E_0\dot{\sigma} \ldots E_k\dot{\sigma} \qquad x\dot{\sigma} = \begin{cases} \sigma(x) & \text{if } \sigma(x) \in \mathsf{Name} \\ x & \text{if } x \notin \mathsf{dom}(\sigma) \\ \bullet & \text{otherwise} \end{cases}$$
$$(x_1, \ldots, x_k)\dot{\sigma} = (x_1, \ldots, x_k)$$
$$\texttt{<}M_1, \ldots, M_k\texttt{>}\dot{\sigma} = \texttt{<}M_1\sigma, \ldots, M_k\sigma\texttt{>}$$

Application of a substitution to messages:

$$(M_0.M_1)\sigma = M_0\sigma.M_1\sigma \qquad F\sigma = \begin{cases} \sigma(F) & \text{if } F = x \in \mathsf{dom}(\sigma) \\ F\dot{\sigma} & \text{otherwise} \end{cases}$$
$$\mathsf{0}\sigma = \mathsf{0}$$

Application of a substitution to processes:

$$\mathsf{0}\sigma = \mathsf{0} \qquad (P \mid Q)\sigma = P\sigma \mid Q\sigma \qquad (!P)\sigma = !P\sigma$$
$$(\nu(x).P)\sigma = \nu(x).P\sigma \quad \text{if } x \notin \mathsf{dom}(\sigma) \cup \mathsf{fn}(\sigma)$$
$$(A.P)\sigma = \begin{cases} \sigma(A)_* P\sigma & \text{if } A = x \in \mathsf{dom}(\sigma) \\ A\dot{\sigma}.P\sigma & \text{if } A \notin \mathsf{dom}(\sigma) \;\&\; \mathsf{bn}(A) \cap (\mathsf{dom}(\sigma) \cup \mathsf{fn}(\sigma)) = \emptyset \end{cases}$$

Figure 2: Application of a substitution to Meta✶ entities.

processes P and Q in parallel, "$\nu(x).P$" behaves as P with private name x (i.e., x in P differs from all names outside P), and finally "$!P$" acts as infinitely many copies of P in parallel ("$P \mid P \mid \cdots$"). Let "." bind more tightly than "$\mid$", and let "ν" bind more tightly than "$\mid$". These constructors have standard properties, e.g., "$\mid$" is commutative, adjacent "ν" can be interchanged, etc. In contrast, the semantics of actions is defined by instantiating Meta✶ (see below).

Free and bound names. All occurrences of the name x in "$\nu(x).P$" are (ν-)bound. When the action A contains an element "$(x_1, \ldots, x_k)$" then all occurrences of the x_i's in "$A.P$" as well as in A on its own are called (input-)bound. An occurrence of x that is not bound is free. The occurrence of a in a^i is bound (resp. free) when this occurrence of a^i is. Bound occurrences of names can be α-converted which can rename a^i only to a^j with j arbitrary, i.e., preserving the basic name part a. Processes that are α-convertible are identified. The set of free names of P is denoted $\mathsf{fn}(P)$ and the set of bound names of the action A is written $\mathsf{bn}(A)$.

Well-scopedness. A process P is *well scoped* when (W1) its input-bound, ν-bound, and free basic names do not overlap, (W2) nested input binders do not bind the same basic name, and (W3) no action contains an input-binding of a basic name more than once. Condition W1 allows simplifications, W2 and W3 are important for type inference, and condition W1 also rules out some processes with an unclear meaning like "$\mathsf{a}^0(\mathsf{a}^0).\mathsf{a}^0.0$". Henceforth, we consider only well scoped processes. Well-scopedness can be achieved by renaming if necessary.

Substitution. A Meta✶ substitution, denoted σ, is a finite function from Name to Message. Fig. 2 defines applying substitutions to Meta✶ entities. This is written postfix as $M\sigma$ and $P\sigma$ for messages and processes, and as $Z\dot{\sigma}$ for other

$$P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad P \mid \mathsf{0} \equiv P$$
$$\mathsf{0} \equiv \mathsf{!0} \qquad \nu(x).\nu(y).P \equiv \nu(y).\nu(x).P \qquad \mathsf{!}P \equiv P \mid \mathsf{!}P$$
$$\frac{x \notin \mathsf{fn}(A) \cup \mathsf{bn}(A)}{A.\nu(x).P \equiv \nu(x).A.P} \qquad \frac{x \notin \mathsf{fn}(P)}{P \mid \nu(x).Q \equiv \nu(x).(P \mid Q)}$$

Figure 3: Structural equivalence of Meta✶.

Meta✶ entities, and binds more tightly than other operators, e.g., $A.P\sigma$ stands for $A.(P\sigma)$. Let $\mathsf{fn}(\sigma)$ be the names in messages in the range of σ. Fig. 2 also defines the message splicing $M_* P$ which discards empty messages $\mathsf{0}$ from M and pushes components of M from right to left onto P (for example $((\mathsf{a}.\mathsf{b}).\mathsf{c})_* P = \mathsf{a}.\mathsf{b}.\mathsf{c}.P$).

Substitution replaces names by messages, but non-name messages are Meta✶ syntax errors at some name positions. For example, substituting "in a" for b in "open b" would yield "open (in a)" which is invalid syntax. In some process calculi, the syntax allows such expressions but they are semantically inert. In Meta✶, substitution places a special name "•" at positions that would otherwise be syntax errors, e.g., the above example substitution yields "open •".

Structural equivalence. The structural equivalence relation $\equiv$ is the smallest equivalence relation that satisfies the rules in Fig. 3 and is congruent with the Meta✶ process constructors. This expresses the following standard properties of parallel composition, name restriction, and replication. Parallel composition is commutative and associative and has $\mathsf{0}$ as its unit. The scope of name restriction can be extruded from name restriction, parallel composition, and actions when there is no binding conflict. Replication implements repetitive behavior. This basic semantics of operators described by structural equivalence is fixed and does not vary with instantiation of Meta✶.

2.2 Instantiations of Meta✶

Instead of fixing the semantics of actions, Meta✶ provides syntax for specifying rewriting rules that give meaning to actions and also defines how these rules yield a rewriting relation on processes. This is how Meta✶ is instantiated to make a particular process calculus.

Process templates and rewriting rules. Fig. 4 presents the rewriting rule syntax. Process templates are used to describe both left and right-hand sides of rewriting rules. Template syntax resembles process syntax except syntax tree leaves can be variables in addition to names. Variables in templates are replaced during rule instantiation by values of appropriate sorts, i.e., name variables range over names, etc. A substitution application template "$\{\mathring{x}_1 := \mathring{s}_1, \ldots, \mathring{x}_k := \mathring{s}_k\}\,\mathring{p}$" describes a substitution to be applied on the right-hand side of some rule. The **rewrite** rules specify ordinary rewriting rules while **active** rules describe rewriting contexts, i.e., positions in processes other than at top-level where rewriting rules are to be applied.

Syntax of Meta✶ *rewriting rules:*

$$
\begin{array}{rrll}
\mathring{x},\mathring{y} \in & \mathsf{NameVar} & ::= \mathtt{\mathring{a}} \mid \mathtt{\mathring{b}} \mid \mathtt{\mathring{c}} \mid \cdots \\
\mathring{m} \in & \mathsf{MessageVar} & ::= \mathtt{\mathring{M}} \mid \mathtt{\mathring{N}} \mid \cdots \\
\mathring{p} \in & \mathsf{ProcessVar} & ::= \mathtt{\mathring{P}} \mid \mathtt{\mathring{Q}} \mid \mathtt{\mathring{R}} \mid \cdots \\
\mathring{s} \in & \mathsf{Substitute} & ::= \mathring{x} \mid \mathring{m} \\
\mathring{E} \in & \mathsf{ElementTempl} & ::= x \mid \mathring{x} \mid (\mathring{x}_1, \ldots, \mathring{x}_k) \mid \texttt{<}\mathring{m}_1, \ldots, \mathring{m}_k\texttt{>} \\
\mathring{A} \in & \mathsf{ActionTempl} & ::= \mathring{E}_0\ \mathring{E}_1 \ldots \mathring{E}_k \\
\mathring{P},\mathring{Q} \in & \mathsf{ProcessTempl} & ::= \mathring{p} \mid \mathring{A}.\mathring{P} \mid \mathtt{0} \mid (\mathring{P} \mid \mathring{Q}) \mid \{\mathring{x}_1 := \mathring{s}_1, \ldots, \mathring{x}_k := \mathring{s}_k\}\,\mathring{p} \\
\mathring{r} \in & \mathsf{Rule} & ::= \mathbf{rewrite}\{\,\mathring{P} \hookrightarrow \mathring{Q}\,\} \mid \mathbf{active}\{\,\mathring{p}\ \mathbf{in}\ \mathring{P}\,\} \\
\mathcal{R} \in & \mathsf{RuleSet} & = \mathcal{P}_{\mathrm{fin}}(\mathsf{Rule})
\end{array}
$$

Semantics of Meta✶ *rewriting rules:*

$$
\frac{\mathbf{rewrite}\{\,\mathring{P} \hookrightarrow \mathring{Q}\,\} \in \mathcal{R} \quad \text{(I1 \& I2)}}{[\![\mathring{P}]\!]_\rho \xrightarrow{\mathcal{R}} [\![\mathring{Q}]\!]_\rho} \qquad \frac{\mathbf{active}\{\,\mathring{p}\ \mathbf{in}\ \mathring{P}\,\} \in \mathcal{R} \quad P \xrightarrow{\mathcal{R}} Q \quad \text{(I1 \& I2)}}{[\![\mathring{P}]\!]_{\rho[\mathring{p}\mapsto P]} \xrightarrow{\mathcal{R}} [\![\mathring{P}]\!]_{\rho[\mathring{p}\mapsto Q]}}
$$

$$
\frac{P \xrightarrow{\mathcal{R}} Q}{P \mid R \xrightarrow{\mathcal{R}} Q \mid R} \qquad \frac{P \xrightarrow{\mathcal{R}} Q \quad x \notin \mathsf{fn}(\mathcal{R})}{\nu(x).P \xrightarrow{\mathcal{R}} \nu(x).Q} \qquad \frac{P' \equiv P \quad P \xrightarrow{\mathcal{R}} Q \quad Q \equiv Q'}{P' \xrightarrow{\mathcal{R}} Q'}
$$

Figure 4: Syntax and semantics of Meta✶ reduction rule descriptions.

An entity instantiation ρ maps name, message, and process variables respectively to names, messages and processes. Applying ρ to $\mathring{P}$, written $[\![\mathring{P}]\!]_\rho$, instantiates the template to make a process by filling in values for variables in $\mathring{P}$ as assigned by ρ. We forbid the name "$\bullet$" as a value of name variables to prevent distinct earlier error results from being treated as the same name.

Given a rewriting rule set $\mathcal{R}$, Fig. 4 defines the rewriting relation $\xrightarrow{\mathcal{R}}$. The additional conditions I1 and I2 as well as the set $\mathsf{fn}(\mathcal{R})$ of all names of $\mathcal{R}$ are described in the following subsection.

Additional restrictions on rewriting rules. It is desirable to rule out rules and inferences that can capture a free name, release a bound name, unleash a nested input-binders, or that can introduce a nesting of previously not nested input-binders. To ensure that the aboves do not happen we need additional syntactic restrictions on rewriting rules, and additional conditions that apply to inference rules of $\xrightarrow{\mathcal{R}}$. This section describes them.

When the action template $\mathring{A}$ contains an element template '$(\mathring{x}_1, \ldots, \mathring{x}_k)$' then all occurrences of the name variables $\mathring{x}_i$'s in '$\mathring{A}.\mathring{P}$' are said to be bound. Also name variables $\mathring{x}_i$'s are said to be bound in '$\{\mathring{x}_1 := \mathring{s}_1, \ldots, \mathring{x}_k := \mathring{s}_k\}\,\mathring{p}$'. Any occurrence of a variable that is not bound is said to be free. The set $\mathsf{bv}(\mathring{P})$ of bound variables of the process template $\mathring{P}$ is the set of all variables with a bound occurrence. Only a name variable can be bound. The set $\mathsf{fv}(\mathring{P})$ of free variables of $\mathring{P}$ is the set of all variables with a free occurrence. This includes all message and process variables from $\mathring{P}$. The set of all variables of $\mathring{P}$ (either free or bound) is denoted $\mathsf{var}(\mathring{P})$. The set $\mathsf{fn}(\mathring{P})$ is the set of free names of $\mathring{P}$ (those element templates $\mathring{E}$ that are names x). For example, given the process template $\mathring{P} = \mathtt{do(\mathring{y}).\mathring{a}[].(out\ \mathring{b}.\mathring{P} \mid \{\mathring{x} := \mathring{M}\}\ \mathring{Q})}$ we have $\mathsf{bv}(\mathring{P}) = \{\mathring{x}, \mathring{y}\}$,

$\mathsf{fv}(\mathring{P}) = \{\mathring{\mathtt{a}}, \mathring{\mathtt{b}}, \mathring{\mathtt{M}}, \mathring{\mathtt{P}}, \mathring{\mathtt{Q}}\}$, and $\mathsf{fn}(\mathring{P}) = \{\mathsf{do}, [], \mathsf{out}\}$. The set $\mathsf{fn}(\mathcal{R})$ of free names of the rule set $\mathcal{R}$ is the union of the sets of free names of all process templates in $\mathcal{R}$.

In this section, we use the metavariable $\mathring{z}$ to range over all template variables, that is, name, message, and process variables. The following definition defines the notion of the scope of a bound name variable and some useful notations.

DEFINITION 2.1. *We say that an occurrence of $\mathring{z}$ in $\mathring{P}$ is* under the scope *of $\mathring{x}$ when $\mathring{P}$ contains either:*

(U1) *$\mathring{A}.\mathring{Q}$ with the given occurrence of $\mathring{z}$ in $\mathring{Q}$, $\mathring{x} \in \mathsf{bv}(\mathring{A})$, or*
(U2) *$\{\!\!\{ \ldots \mathring{x} := \mathring{s} \ldots \}\!\!\}\, \mathring{p}$ with $\mathring{p} = \mathring{z}$ being the given occurrence of $\mathring{z}$.*

Write $\mathring{P} \vdash_\exists \mathring{x} \gg \mathring{z}$ when there is an occurrence of $\mathring{z}$ in $\mathring{P}$ under the scope of $\mathring{x}$.
Write $\mathring{P} \vdash_\forall \mathring{x} \gg \mathring{z}$ when all occurrences of $\mathring{z}$ in $\mathring{P}$ are under the scope of $\mathring{x}$.

The following defines defines additional restrictions that applies to left hand side templates in rewriting rules.

DEFINITION 2.2. *We say that $\mathring{P}$ is a* well formed lhs-template *when $\mathring{P}$ satisfies the following properties:*

(L1) $\mathsf{fv}(\mathring{P}) \cap \mathsf{bv}(\mathring{P}) = \emptyset$
(L2) *any message and process variable occurs at most once in $\mathring{P}$*
(L3) *$\mathring{P}$ does not contain $\{\!\!\{\mathring{x}_1 := \mathring{s}_1, \ldots, \mathring{x}_k := \mathring{s}_k\}\!\!\}\, \mathring{p}$*
(L4) *when $\mathring{P}$ contains $\mathring{A}$ then every $\mathring{x} \in \mathsf{bv}(\mathring{A})$ occurs exactly once in $\mathring{A}$*
(L5) *when $\mathring{P} \vdash_\exists \mathring{x} \gg \mathring{z}$ then $\mathring{P} \vdash_\forall \mathring{x} \gg \mathring{z}$*

Similarly the following restrictions apply to the right hand side templates in a rewriting rule.

DEFINITION 2.3. *We say that $\mathring{Q}$ is a* well formed rhs-template *w.r.t. a well formed lhs-template $\mathring{P}$ when $\mathring{Q}$ satisfies the following properties:*

(R1) $\mathsf{fv}(\mathring{Q}) \subseteq \mathsf{fv}(\mathring{P})$
(R2) $\mathsf{bv}(\mathring{Q}) \subseteq \mathsf{bv}(\mathring{P})$
(R3) *when $\mathring{Q}$ contains $\{\!\!\{\mathring{x}_1 := \mathring{s}_1, \ldots, \mathring{x}_k := \mathring{s}_k\}\!\!\}$ then $\mathring{x}_i$'s are pairwise distinct*
(R4) *when $\mathring{Q}$ contains $\mathring{A}$ then every $\mathring{x} \in \mathsf{bv}(\mathring{A})$ occurs exactly once in $\mathring{A}$*
(R5) *when $\mathring{Q} \vdash_\exists \mathring{x} \gg \mathring{z}$ then $\mathring{Q} \vdash_\forall \mathring{x} \gg \mathring{z}$*
(R6) *for $\mathring{z} \in \mathsf{var}(\mathring{Q})$ and any $\mathring{x}$ holds that $\mathring{P} \vdash_\forall \mathring{x} \gg \mathring{z}$ iff $\mathring{Q} \vdash_\forall \mathring{x} \gg \mathring{z}$*

The following introduces the notion of a well formed rewriting rule.

DEFINITION 2.4. *The rule* **rewrite**$\{\!\!\{ \mathring{P} \hookrightarrow \mathring{Q} \}\!\!\}$ *is said to be* well formed *when $\mathring{P}$ is a well formed lhs-template and $\mathring{Q}$ is a well formed rhs-template w.r.t. $\mathring{P}$. The rule* **active**$\{\!\!\{ \mathring{p}$ **in** $\mathring{P} \}\!\!\}$ *is said to be* well formed *when $\mathring{P}$ is a well formed lhs-template. The rule set $\mathcal{R}$ is called a* well formed rule set, *written $\mathsf{wf}(\mathcal{R})$, when all of its rules are well formed.*

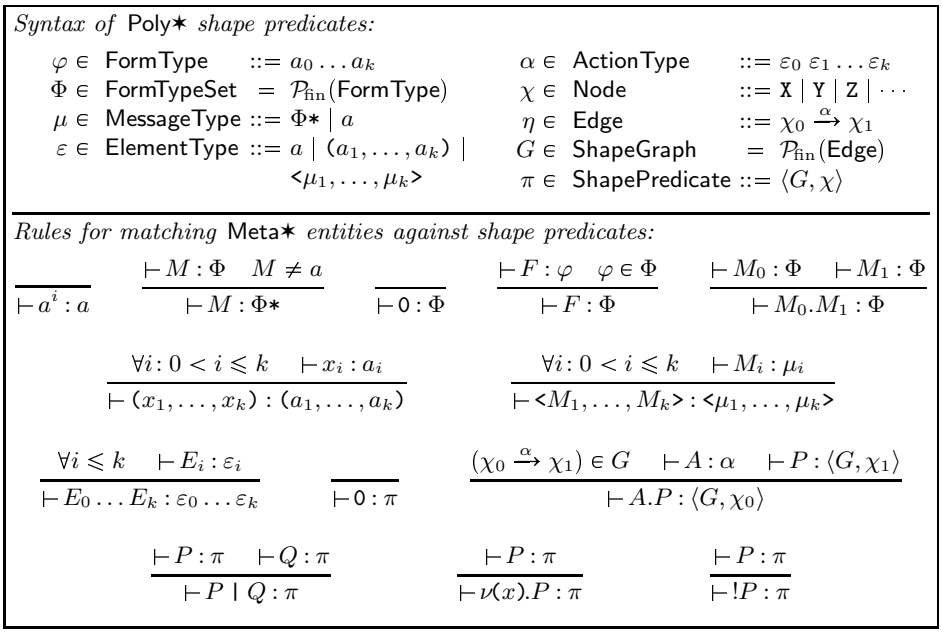


Figure 5: Syntax and semantics of Poly✶ shape predicates.

From now on we suppose only well formed rule sets. Alternatively we could add the condition $\mathsf{wf}(\mathcal{R})$ to the premise of rule RRw. Additionally we require the following condition to be satisfied for both RRw and RAct to avoid name captures when picking a name representant for input-binders:

(I1) whenever $\mathring{x}, \mathring{y} \in \mathsf{bv}(\mathring{P})$ and $\mathring{x} \neq \mathring{y}$ then $[\![\mathring{x}]\!]_\rho \neq [\![\mathring{y}]\!]_\rho$, and
(I2) whenever $\mathring{x} \in \mathsf{bv}(\mathring{P})$ then $[\![\mathring{x}]\!]_\rho \notin \mathsf{fn}(\mathcal{R})$.

2.3 Poly✶ Shape Predicates and Types for Meta✶

Shape Predicates. Poly✶ types are built on the notion of *shape predicates*. A *shape predicate* describes possible structures of process syntax trees. A shape predicate's meaning is the set of all processes with the given structure. When a rewriting rule from $\mathcal{R}$ is applied to a process, its syntax tree changes, and sometimes the new syntax tree is no longer satisfies the same shape predicates. All Poly✶ ($\mathcal{R}$-)types are shape predicates that describe process sets closed under rewriting using $\mathcal{R}$. For feasibility, types are defined via a syntactic test that enforces rewriting-closedness. Further restrictions are used to ensure the existence of principal typings.

Fig. 5 defines the syntax of shape predicates. Action types describe actions and have corresponding syntax. The main difference between actions and action types are that actions are built from basic names instead of names, and that compound messages are described up to commutativity, associativity, and repetitions of their parts. Thus a single action type describes a set of actions.

A shape predicate is a rooted directed finite graph with edges labeled by action types. A process matches a shape predicate when the process's syntax tree is a "subgraph" of the shape predicate. Because a shape predicate can have loops, it can describe syntax trees of arbitrary height.

Fig. 5 also describes matching Meta✶ entities against types. The rule matching actions against action types also matches forms against form types. Matching processes against shape predicates is independent of rewriting rules, i.e., it works the same in any Meta✶ instantiation. The *meaning* of the shape predicate π, written $[\![\pi]\!]$, is the set of all processes matching π, namely $\{P : \vdash P : \pi\}$.

2.4 Poly✶ Types and Syntactically Closed Shape Predicates

This section provides details about syntactically closed shape predicates regarded as Poly✶ types.

Semantically Closed Shape Predicates. A shape predicate π is *semantically closed* w.r.t. rewriting rule set $\mathcal{R}$ when the meaning of π is closed under $\mathcal{R}$-rewritings. Formally as follows.

Definition 2.5. *Let $\mathcal{R}$ be a rule set. We call the shape predicate π* semantically closed *w.r.t. $\mathcal{R}$, written $\mathcal{R} \bullet\!\Rightarrow \pi$, iff*

$$\vdash P : \pi \quad \textit{and} \quad P \overset{\mathcal{R}}{\hookrightarrow} Q \qquad \textit{imply} \qquad \vdash Q : \pi$$

Because deciding if a shape predicate is semantically closed w.r.t. an arbitrary $\mathcal{R}$ is nontrivial, we use an easier-to-decide property of shape predicates, namely *syntactic closure*, which by design is algorithmically verifiable. The following sections lead to the definition of this notion.

Type Substitutions and Flow Edges. Shape graphs also contain *flow edges*, which are used in type inference algorithms and in recognizing syntactic closure. While the action edges shown in Fig. 5 are labeled with action types, flow edges are labeled instead with *type substitutions*, finite functions from basic names to message types. A type substitution τ represents a set of Meta✶ substitutions.

Definition 2.6. *The* type substitution *τ is defined below. We extend the definition of shape predicates from Fig. 5 as follows:*

$$\begin{array}{lll} \tau \in \mathsf{TypeSubstitution} & = & \mathsf{Name} \xrightarrow{\text{fin}} \mathsf{MessageType} \\ \eta \in \mathsf{Edge} & ::= & \cdots \mid \chi_0 \xrightarrow{\boxed{\tau}} \chi_1 \end{array}$$

Application of a type substitution to Meta✶ form types, message types, and element types is defined in Fig. 6. The postfix application $\dot{\tau}$ maps form types to sets of form types, $\ddot{\tau}$ maps message types to message types, and the postfix application of τ maps element types to element types. Note that as a special case the postfix application of τ maps basic names to basic names.

Application of a type substitution to form types $\dot{\tau}$:

$$(a_0 \dots a_k)\dot{\tau} = \begin{cases} \Phi & \text{if } k = 0 \;\&\; \tau(a_0) = \Phi* \\ \{(a_0\tau) \dots (a_k\tau)\} & \text{otherwise} \end{cases}$$

Application of a type substitution to message types $\ddot{\tau}$:

$$a\ddot{\tau} = \begin{cases} \tau(a) & \text{if } a \in \mathsf{dom}(\tau) \\ a & \text{otherwise} \end{cases}$$

$$(\{\varphi_1, \dots, \varphi_k\}*)\ddot{\tau} = (\varphi_1\dot{\tau} \cup \dots \cup \varphi_k\dot{\tau})*$$

Application of a type substitution to element types and actions:

$$a\tau = \begin{cases} \tau(a) & \text{if } \tau(a) \in \mathsf{Name} \\ a & \text{if } a \notin \mathsf{dom}(\tau) \\ \bullet & \text{otherwise} \end{cases}$$

$$<\mu_1, \dots, \mu_k>\tau = <\mu_1\ddot{\tau}, \dots, \mu_k\ddot{\tau}>$$

$$(a_1, \dots, a_k)\tau = (a_1, \dots, a_k)$$

$$(\varepsilon_0 \dots \varepsilon_k)\tau = (\varepsilon_0\tau) \dots (\varepsilon_k\tau)$$

Figure 6: Application of a type substitution to form types, message types, and element types.

A flow edge $(\chi \multimap\!\!\boxed{\tau}\!\!\to \chi') \in G$ expresses the property that whenever the process P matches $\langle G, \chi\rangle$ and σ is a substitution represented by τ, then the process $P\sigma$ must match $\langle G, \chi'\rangle$. Flow edges describe possible movements of processes that involve substitution application. Syntactic closure insists on the presence of flow edges implied by rewriting rules or other flow edges. Of course the above property associtated with a flow edge is not satisfied automatically.

The Meaning of Flow Edges. A shape graph is flow closed when the intuitive meaning of flow edges described in the previous paragraph is satisfied. Note that while the existence of particular flow edges is involved by the set of rewriting rules $\mathcal{R}$, the notion of flow-closedness itself does not depend on $\mathcal{R}$. Bound basic names of the action type α, denoted $\mathsf{bn}(\alpha)$, are those basic names of α that appear inside some input element type $(a_1, \dots, a_k)$.

Definition 2.7. *The shape graph G is said to be* flow-closed *iff whenever it contains $\chi \xrightarrow{\alpha} \chi'$ and $\chi \multimap\!\!\boxed{\tau}\!\!\to \chi_0$ such that $\mathsf{bn}(\alpha) \cap \mathsf{dom}(\tau) = \emptyset$ then it holds that*

(F1) *if $\tau(\alpha) = \{\varphi_1, \dots, \varphi_k\}*$ then $\{\chi_0 \xrightarrow{\varphi_i} \chi_0 : 0 < i \leqslant k\} \cup \{\chi' \multimap\!\!\boxed{\tau}\!\!\to \chi_0\} \subseteq G$*
(F2) *otherwise there is χ'_0 such that $\{\chi' \multimap\!\!\boxed{\tau}\!\!\to \chi'_0, \chi_0 \xrightarrow{\alpha\tau} \chi'_0\} \subseteq G$.*

We call the shape predicate $\langle G, \chi\rangle$ flow-closed iff its G component is.

Syntactically Closed Shape Predicates. Type instantiations are used to relate process templates and shapes graph just like entity instantiations are used to relate templates and processes. However, the below defined relation between templates and shape graphs is not functional like in the case of templates and processes, where an entity instantiation and a template unambigiously determine a process. Rather, it relates a type instantiation and a template with several shape predicates. The formal definition is as follows.

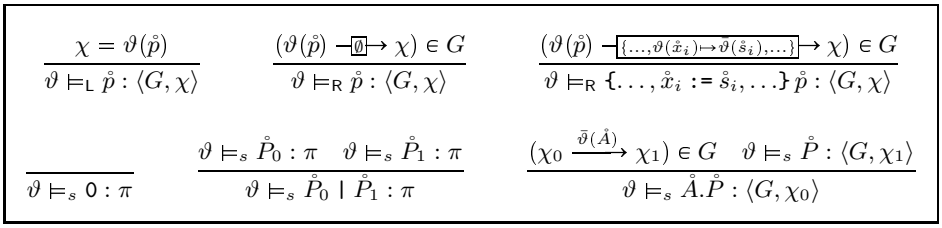


Figure 7: Matching of process templates to shape graphs. The rules for template processes have an L variant and an R variant; the variable letter s ranges over L and R.

DEFINITION 2.8. *A* type instantiation ϑ *is a finite function mapping* NameVar *to* BasicName$\backslash\{\bullet\}$, MessageVar *to* MessageType, *and* ProcessVar *to* Node. *Let* $\bar{\vartheta}$ *denote application of* ϑ *to* Meta$\star$ *element templates, form templates, and to substitutes. It maps element templates to element types, action templates to action types, and substitutes to message types.*

$$\begin{array}{ll} \bar{\vartheta}(a^i) = a & \bar{\vartheta}(\mathring{x}_1, \ldots, \mathring{x}_k) = (\vartheta(\mathring{x}_1), \ldots, \vartheta(\mathring{x}_k)) \\ \bar{\vartheta}(\mathring{x}) = \vartheta(\mathring{x}) & \bar{\vartheta}(\langle \mathring{m}_1, \ldots, \mathring{m}_k \rangle) = \langle \vartheta(\mathring{m}_1), \ldots, \vartheta(\mathring{m}_k) \rangle \\ \bar{\vartheta}(\mathring{m}) = \vartheta(\mathring{m}) & \bar{\vartheta}(\mathring{E}_0 \ldots \mathring{E}_k) = \bar{\vartheta}(\mathring{E}_0) \ldots \bar{\vartheta}(\mathring{E}_k) \end{array}$$

The relation between type instantiations and process templates is given by the inference system in Fig. 7. As a special exception, $\vartheta \models_s \mathring{P} : \pi$ *is* not *considered to hold if* $\vartheta(\mathring{x}_0) = \vartheta(\mathring{x}_1)$ *for* $\mathring{x}_0 \neq \mathring{x}_1$ *such that* $\mathring{x}_0$ *occurs in* $\mathring{P}$ *below a form template containing a binding element* $(\ldots, \mathring{x}_1, \ldots)$.

The relation $\models_{\mathsf{L}}$ is used to relate a left-hand-side template with a shape graph. Informally, $\vartheta \models_{\mathsf{L}} \mathring{P} : \pi$ says that $\mathring{P}$ can be instatiated to some process that matches π and that ϑ describes this instatiation. Alternatively, the above statement says that $\mathring{P}$ can be "attached" to π when variables in $\mathring{P}$ are filled in accrodingly to ϑ. Similarly, $\models_{\mathsf{R}}$ is used to relate right-hand-side templates with shape graphs. It differs in that it takes existening flow edges into account and it allows the template to contain substitution constructions. Suppose the rewriting rule $\textbf{rewrite}\{\, \mathring{P} \hookrightarrow \mathring{Q} \,\}$ is given and that $\vartheta \models_{\mathsf{L}} \mathring{P} : \pi$ holds. Then the shape graph π has to contain certain additional edges in order for $\vartheta \models_{\mathsf{R}} \mathring{Q} : \pi$ to hold. In this way these two relations are used to force the existence of edges important for the meaning of π to be closed under the rewriting rule.

The following defines the set of active nodes determined by **active** rules, that is, the set of nodes where rewritings rules are to be applied.

DEFINITION 2.9. *Let the shape predicate* $\pi = \langle G, \chi_0 \rangle$ *be given. The set of* active nodes *for* $\mathcal{R}$, *written* $\mathsf{active}(\pi, \mathcal{R})$, *is the least set* A *of nodes which contains* X *and such that for all* $Y \in A$ *and all* $\textbf{active}\{\, \mathring{p} \textbf{ in } \mathring{P} \,\} \in \mathcal{R}$, *it holds that* $\vartheta \models_{\mathsf{L}} \mathring{P} : \langle G, \chi_1 \rangle$ *implies* $\vartheta(\mathring{p}) \in A$.

A shape predicate is locally closed at some node when application of any of rewriting rules at the node does not insist the existence of a new flow edge.

DEFINITION 2.10. *G is* locally closed *at χ w.r.t. $\mathcal{R}$ iff whenever $\mathcal{R}$ contains the rule* $\mathbf{rewrite}\{\, \mathring{P} \hookrightarrow \mathring{Q} \,\}$ *it holds that $\vartheta \vDash_{\mathsf{L}} \mathring{P} : \langle G, \chi \rangle$ implies $\vartheta \vDash_{\mathsf{R}} \mathring{Q} : \langle G, \chi \rangle$.*

Finally, syntactically closed shape predicates are those both flow closed and locally closed at any active node.

DEFINITION 2.11. *The shape predicate π is* syntactically closed *w.r.t. $\mathcal{R}$ iff G is flow-closed and also locally closed w.r.t. $\mathcal{R}$ at every $\chi \in \mathsf{active}(\pi, \mathcal{R})$. When this holds, we call π an $\mathcal{R}$-*type*, denoted by $\mathcal{R} \bullet\!\!\rightarrow \pi$. When $\mathcal{R} \bullet\!\!\rightarrow \pi$ and $\vdash P : \pi$ we say that π is an $\mathcal{R}$-type of P.*

Subject reduction says that the meaning of a syntactically closed shape predicate is closed under rewritings.

THEOREM 2.12 (SUBJECT REDUCTION). *For every $\pi \in \mathsf{ShapePredicate}$ and $\mathcal{R} \in \mathsf{RuleSet}$, it holds that $\mathcal{R} \bullet\!\!\rightarrow \pi$ implies $\mathcal{R} \bullet\!\!\Rightarrow \pi$.*

Poly✶ Principal Typings. A type π of P is a *principal typing* of P when $[\![\pi]\!] \subseteq [\![\pi_0]\!]$ for any other type π_0 of P. The following section defines two properties of shape graphs called the *width* and the *depth* restriction. Among Poly✶ types satisfying these restriction the existence of the principal typing for every P hold.

At first we define the binary relation $\approx$ on action types as follows.

DEFINITION 2.13. *Write $\alpha_0 \approx \alpha_1$ iff there is A such that $\vdash A : \alpha_0$ and $\vdash A : \alpha_1$.*

The $\approx$ relation is close to being the equality on action types. The only way for non-identical α's to be related by $\approx$ is when one of them contains some message type $\Phi*$. It is relatively safe to image $\approx$ to be $=$, at least to the first approximation. It is necessary to take this relation instead of $=$ in two definitions below in order to achieve the principal typing property. Definitions of the width and depth restriction on shape graphs follow. A shape predicate is said to satisfy one of these properties if its shape graph component satisfies the property.

DEFINITION 2.14. *$G \in \mathsf{ShapeGraph}$ satisfies the* width restriction *iff whenever there are two edges $(\chi \xrightarrow{\alpha} \chi_0) \in G$ and $(\chi \xrightarrow{\alpha'} \chi_1) \in G$ with $\alpha \approx \alpha'$, then it holds that $\chi_0 = \chi_1$.*

DEFINITION 2.15. *$G \in \mathsf{ShapeGraph}$ satisfies the* depth restriction *iff whenever there is a path of edges in G like $\chi_0 \xrightarrow{\alpha_1} \chi_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_k} \chi_k$ with $\alpha_1 \approx \alpha_k$, then it holds that $\chi_1 = \chi_k$.*

Among syntacticly closed shape predicates that satisfy the width and depth restrictions the principal typing property can be proved [MW04]. Moreover, the type inference algorithm is implemented.

3 Poly✶ and Types for Mobile Ambients

This section compares the type system of TMA, a typed version of MA by Cardelli and Gordon, with the instantiation of Poly✶ to the terms and rewriting rules of TMA. Sec. 3.1 describes TMA, Sec. 3.2 instantiates Meta✶ to TMA and describe some properties, Sec. 3.3 shows how to embed TMA predicates in Poly✶ types, and Sec. 3.4 discusses possible extensions of the embedding.

3.1 Types for Mobile Ambients (TMA)

Mobile Ambients (MA), introduced by Cardelli and Gordon [CG98], is a process calculus for representing process mobility. Processes are placed inside named bounded locations called *ambients* which form a tree hierarchy. Processes can change the hierarchy. Processes can also send to nearby processes messages containing either either ambient names or hierarchy change instructions. A typed version of MA [CG99] introduced by Cardelli and Gordon assigns an allowed communication topic to each ambient location. and ensures that communications respect the allowed topic at each location.

We compare Poly✶ and a minor modification of the above-mentioned typed version of MA which we call TMA, which differs from Cardelli and Gordon's system in two minor details, allowing a simpler presentation that avoids irrelevant technical issues. First, processes are built from Meta✶ names, that is, from pairs of basic names and integers, and α-conversion works as in Meta✶. Second, TMA has a well-scopedness requirement analogous to that of Meta✶. The formal equivalence of TMA and the original system is obvious.

Fig. 8 describes TMA process syntax. Some names, e.g., "in", are reserved for the translation. Capabilities are ambient hierarchy change instructions. Executing a capability causes the surrounding ambient to participate in a change. The capability "in n" instructs the surrounding ambient to move itself and its contents into a sibling ambient named n. Similarly, "out n" instructs the surrounding ambient to move out of a parent ambient named n and become its sibling. The capability "open n" instructs the surrounding ambient to dissolve the boundary of a child ambient named n. Although the syntax allows an arbitrary N at the position n so that substituting a capability for a name yields valid syntax, capabilities where N is not a single name are inert and meaningless. In capability sequences, the left-most capability will be executed first. Executing a capability consumes it.

The process constructors "0", "|", ".", "!", and "ν" have standard meanings. The name n is (ν-)bound in $(\nu n : W)B$ and comes with an explicit type annotation. Types are described below. The expression $n[B]$ describes the process B running inside the ambient n. As above, the syntax allows inert meaningless constructions with N at the position of n. Capabilities can be communicated in messages. $\langle N_1, \ldots, N_k \rangle$ is a process that sends a k-tuple of messages. $(n_1 : W_1, \ldots, n_k : W_k).B$ is a process that receives a k-tuple of messages, substitutes them for appropriate n's in B, and continues as this new process. The name n_i is said to be (input-)bound in the process and, again, comes with an

Syntax of TMA *processes:*

$$
\begin{array}{rrcl}
n \in & \mathsf{AName} & = & \mathsf{Name} \backslash \{\bullet, \mathsf{in}, \mathsf{out}, \mathsf{open}, \texttt{[]}\} \\
N \in & \mathsf{ACapability} & ::= & \varepsilon \mid n \mid \mathsf{in}\ N \mid \mathsf{out}\ N \mid \mathsf{open}\ N \mid N.N' \\
W \in & \mathsf{AMessageType} & ::= & \text{definition postponed to Fig. 9} \\
B \in & \mathsf{AProcess} & ::= & \mathsf{0} \mid (B_0 \mid B_1) \mid N\texttt{[}B\texttt{]} \mid N.B \mid !B \mid (\nu n{:}W)B \mid \\
 & & & \texttt{<}N_1, \ldots, N_k\texttt{>} \mid \texttt{(}n_1{:}W_1, \ldots, n_k{:}W_k\texttt{)}.B
\end{array}
$$

Structural equivalence of TMA*:*

$$
\begin{array}{llll}
\varepsilon.B \equiv B & B \mid \mathsf{0} \equiv B & B_0 \mid B_1 \equiv B_1 \mid B_0 & B_0 \mid (B_1 \mid B_2) \equiv (B_0 \mid B_1) \mid B_2 \\
!\mathsf{0} \equiv \mathsf{0} & !B \equiv B \mid !B & (N.N').B \equiv N.(N'.B) & (\nu n{:}\mathsf{Amb}\texttt{[}T\texttt{]})\mathsf{0} \equiv \mathsf{0}
\end{array}
$$

$$
\frac{n \neq m}{(\nu n{:}W)(m\texttt{[}B\texttt{]}) \equiv m\texttt{[}(\nu n{:}W)B\texttt{]}} \qquad \frac{n \notin \mathsf{fn}(B_0)}{B_0 \mid (\nu n{:}W)B_1 \equiv (\nu n{:}W)(B_0 \mid B_1)}
$$

$$
\frac{n \neq m}{(\nu n{:}W_0)(\nu m{:}W_1)B \equiv (\nu m{:}W_1)(\nu n{:}W_0)B}
$$

Rewriting relation of TMA*:*

$$
\begin{array}{rcl}
n\texttt{[}\mathsf{in}\ m.B_0 \mid B_1\texttt{]} \mid m\texttt{[}B_2\texttt{]} & \rightarrow & m\texttt{[}n\texttt{[}B_0 \mid B_1\texttt{]} \mid B_2\texttt{]} \\
m\texttt{[}n\texttt{[}\mathsf{out}\ m.B_0 \mid B_1\texttt{]} \mid B_2\texttt{]} & \rightarrow & n\texttt{[}B_0 \mid B_1\texttt{]} \mid m\texttt{[}B_2\texttt{]} \\
\mathsf{open}\ n.B_0 \mid n\texttt{[}B_1\texttt{]} & \rightarrow & B_0 \mid B_1 \\
\texttt{(}n_1{:}W_1, \ldots, n_k{:}W_k\texttt{)}.B \mid \texttt{<}N_1, \ldots, N_k\texttt{>} & \rightarrow & B\{n_1 \mapsto N_1, \ldots, n_k \mapsto N_k\}
\end{array}
$$

$$
\frac{B_0 \rightarrow B_1}{(\nu n{:}W)B_0 \rightarrow (\nu n{:}W)B_1} \qquad \frac{B_0 \rightarrow B_1}{n\texttt{[}B_0\texttt{]} \rightarrow n\texttt{[}B_1\texttt{]}} \qquad \frac{B_0 \rightarrow B_1}{B_0 \mid B_2 \rightarrow B_1 \mid B_2}
$$

$$
\frac{B_0' \equiv B_0 \quad B_0 \rightarrow B_1 \quad B_1 \equiv B_1'}{B_0' \rightarrow B_1'}
$$

Figure 8: Syntax and semantics of TMA.

explicit type annotation. Bound basic names and the free names of a process are defined like in Meta✶. Processes that are α-convertible are identified. Fig. 8 also describes structural equivalence and semantics of TMA processes. The only thing the semantics does with type annotations is copy them around.

Fig. 9 describes TMA type syntax. Exchange types are assigned to processes and ambient locations to describe allowed communication topics. The type Shh indicates silence (no communication). $W_1 \otimes \cdots \otimes W_k$ indicates communication of k-tuples of messages whose i-th member has the message type W_i. When $k = 0$ we write $\mathbf{1}$ which describes processes executing only synchronization actions `<>` and `()`. Message types describe messages (capability sequences and names). Amb[T] is the type of the name of an ambient where communication described by T is allowed. Cap[T] describes capabilities whose execution can unleash communication described by T (by opening some ambient). Environments assign message types to free names. Fig. 9 also describes the TMA typing rules. (For more details see [CG99].)

TMA uses well-scopedness rules similar to those of Meta✶. A process B is *well scoped* when (S1) its input-bound, ν-bound, and free basic names do not

Syntax of TMA *types:*

$$W \in \mathsf{AMessageType} ::= \mathsf{Amb}[T] \mid \mathsf{Cap}[T]$$
$$T \in \mathsf{AExchangeType} ::= \mathsf{Shh} \mid W_1 \otimes \cdots \otimes W_k$$
$$E \in \mathsf{AEnvironment} = \mathsf{AName} \xrightarrow{\text{fin}} \mathsf{AMessageType}$$

Typing rules of TMA*:*

$$\frac{E(n) = W}{E \vdash n : W} \qquad \frac{}{E \vdash \varepsilon : \mathsf{Cap}[T]} \qquad \frac{E \vdash N : \mathsf{Cap}[T] \quad E \vdash N' : \mathsf{Cap}[T]}{E \vdash N.N' : \mathsf{Cap}[T]}$$

$$\frac{E \vdash N : \mathsf{Amb}[T']}{E \vdash \mathsf{in}\ N : \mathsf{Cap}[T]} \qquad \frac{E \vdash N : \mathsf{Amb}[T']}{E \vdash \mathsf{out}\ N : \mathsf{Cap}[T]} \qquad \frac{E \vdash N : \mathsf{Amb}[T]}{E \vdash \mathsf{open}\ N : \mathsf{Cap}[T]}$$

$$\frac{E \vdash B : T}{E \vdash !B : T} \qquad \frac{E \vdash N : \mathsf{Cap}[T] \quad E \vdash B : T}{E \vdash N.B : T} \qquad \frac{E \vdash N : \mathsf{Amb}[T] \quad E \vdash B : T}{E \vdash N[B] : T'}$$

$$\frac{E \vdash B_0 : T \quad E \vdash B_1 : T}{E \vdash B_0 \mid B_1 : T} \qquad \frac{}{E \vdash 0 : T} \qquad \frac{\forall i : 0 < i \leqslant k \quad E \vdash N_i : W_i}{E \vdash \langle N_1, \ldots, N_k \rangle : W_1 \otimes \cdots \otimes W_k}$$

$$\frac{E[n_1 \mapsto W_1, \ldots, n_k \mapsto W_k] \vdash B : W_1 \otimes \cdots \otimes W_k}{E \vdash (n_1 : W_1, \ldots, n_k : W_k).B : W_1 \otimes \cdots \otimes W_k} \qquad \frac{E[n \mapsto \mathsf{Amb}[T']] \vdash B : T}{E \vdash (\nu n : \mathsf{Amb}[T'])B : T}$$

Figure 9: Syntax of TMA types and typing rules.

overlap, (S2) nested input binders do not bind the same basic name, and (S3) the same type is assigned to bound names with the same basic name. The environment E is *well scoped* when (S4) it assigns the same type to names with the same basic name. Henceforth, we require processes and environments to be well scoped. (Well-scopedness can be achieved by renaming if necessary.)

Definition 3.1. *Call B or E* well formed *when all of the following hold:*

(S1) $\mathsf{fbn}(B)$, $\mathsf{ibbn}(B)$, *and* $\mathsf{nbbn}(B)$ *are pairwise disjoint*
(S2) *for* $(a_1^{i_1} : W_1, \ldots, a_k^{i_k} : W_k).B_0$ *in* B, a_j*'s are distinct and* $a_j \notin \mathsf{ibbn}(B_0)$
(S3) *different binding occurrences of "a" assign the same type to "a"*
(S4) E *assigns the same type to names which share a basic name*

Example 3.2. *In this* TMA *process, packet ambient* p *delivers a synchronization message to destination ambient* d *following instructions* x*. Note that* $E \vdash B : T$*.*

$$E = \{\mathsf{d} \mapsto \mathsf{Amb}[\mathbf{1}]\} \qquad T = \mathsf{Cap}[\mathbf{1}]$$
$$B = \langle \mathsf{in\ d} \rangle \mid (\nu \mathsf{p} : \mathsf{Amb}[\mathbf{1}])(\mathsf{d}[\mathsf{open\ p}.0] \mid (\mathsf{x} : \mathsf{Cap}[\mathbf{1}]).\mathsf{p}[\mathsf{x}.\langle\rangle])$$

This example will also demonstrate the type embedding construction in Sec. 3.3.

$$\overline{N} = \begin{cases} n & \text{if } N = n \in \mathsf{AName} \\ \bullet & \text{otherwise} \end{cases} \qquad \begin{array}{l} (\!|\mathsf{in}\ N|\!) = \mathsf{in}\ \overline{N} \\ (\!|\mathsf{out}\ N|\!) = \mathsf{out}\ \overline{N} \\ (\!|\mathsf{open}\ N|\!) = \mathsf{open}\ \overline{N} \end{array} \qquad \begin{array}{l} (\!|\varepsilon|\!) = \mathtt{0} \\ (\!|N_0.N_1|\!) = (\!|N_0|\!).(\!|N_1|\!) \end{array}$$

$$\begin{array}{ll} (\!|\mathtt{0}|\!) = \mathtt{0} & (\!|B_0 \mid B_1|\!) = (\!|B_0|\!) \mid (\!|B_1|\!) \\ (\!|!B|\!) = !(\!|B|\!) & (\!|\mathtt{<}N_1, \ldots, N_k\mathtt{>}|\!) = \mathtt{<}(\!|N_1|\!), \ldots, (\!|N_k|\!)\mathtt{>.0} \\ (\!|N.B|\!) = (\!|N|\!)_*(\!|B|\!) & (\!|(\nu x : W)B|\!) = \nu(x).(\!|B|\!) \\ (\!|N\mathtt{[}B\mathtt{]}|\!) = \overline{N}\mathtt{[}(\!|B|\!)\mathtt{]} & (\!|(x_1 : W_1, \ldots, x_k : W_k).B|\!) = (x_1, \ldots, x_k).(\!|B|\!) \end{array}$$

Figure 10: Encoding of TMA processes in Meta✶.

3.2 The Instantiation of Meta✶ to TMA

Meta✶ is instantiated to TMA using TMA's rewriting relation written as follows in the syntax from Fig. 4:

$$\begin{array}{rl} \mathcal{A} = \{ & \mathbf{active}\{\mathring{\mathtt{P}}\ \mathbf{in}\ \mathring{\mathtt{a}}\mathtt{[]}.\mathring{\mathtt{P}}\}, \\ & \mathbf{rewrite}\{\mathring{\mathtt{a}}\mathtt{[in}\ \mathring{\mathtt{b}}.\mathring{\mathtt{P}} \mid \mathring{\mathtt{Q}}\mathtt{]} \mid \mathring{\mathtt{b}}\mathtt{[}\mathring{\mathtt{R}}\mathtt{]} \hookrightarrow \mathring{\mathtt{b}}\mathtt{[}\mathring{\mathtt{a}}\mathtt{[}\mathring{\mathtt{P}} \mid \mathring{\mathtt{Q}}\mathtt{]} \mid \mathring{\mathtt{R}}\mathtt{]}\}, \\ & \mathbf{rewrite}\{\mathring{\mathtt{a}}\mathtt{[}\mathring{\mathtt{b}}\mathtt{[out}\ \mathring{\mathtt{a}}.\mathring{\mathtt{P}} \mid \mathring{\mathtt{Q}}\mathtt{]} \mid \mathring{\mathtt{R}}\mathtt{]} \hookrightarrow \mathring{\mathtt{a}}\mathtt{[}\mathring{\mathtt{R}}\mathtt{]} \mid \mathring{\mathtt{b}}\mathtt{[}\mathring{\mathtt{P}} \mid \mathring{\mathtt{Q}}\mathtt{]}\}, \\ & \mathbf{rewrite}\{\mathtt{open}\ \mathring{\mathtt{a}}.\mathring{\mathtt{P}} \mid \mathring{\mathtt{a}}\mathtt{[}\mathring{\mathtt{R}}\mathtt{]} \hookrightarrow \mathring{\mathtt{P}} \mid \mathring{\mathtt{R}}\} \} \cup \\ \bigcup_{k=0}^{\infty} \{ & \mathbf{rewrite}\{\mathtt{<}\mathring{\mathtt{M}}_1, \ldots, \mathring{\mathtt{M}}_k\mathtt{>}.\mathring{\mathtt{P}} \mid (\mathring{\mathtt{a}}_1, \ldots, \mathring{\mathtt{a}}_k).\mathring{\mathtt{Q}} \hookrightarrow \mathring{\mathtt{P}} \mid \{\mathring{\mathtt{a}}_1 := \mathring{\mathtt{M}}_1, \ldots, \mathring{\mathtt{a}}_k := \mathring{\mathtt{M}}_k\}\mathring{\mathtt{Q}}\} \} \end{array}$$

This straightforwardly translates TMA's rules. The **active** rule lets rewriting be done inside ambients and corresponds to the TMA rule '$B_0 \to B_1 \Rightarrow n\mathtt{[}B_0\mathtt{]} \to n\mathtt{[}B_1\mathtt{]}$'. Each communication prefix length has its own rule; in our implementation, a single rule can uniformly handle all lengths, but the formal Meta✶ presentation is deliberately simpler.

The set $\mathcal{A}$ specifies the rewriting relation $\overset{\mathcal{A}}{\hookrightarrow}$. To compare $\overset{\mathcal{A}}{\hookrightarrow}$ with TMA's relation we encode TMA processes into Meta✶ processes. This encoding, presented in Fig. 10, is straightforward due to the flexibility of Meta✶ syntax. The encoding $(\!|\cdot|\!)$ translates capabilities to Meta✶ messages and TMA processes to Meta✶ processes. Meaningless expressions allowed by TMA's syntax are translated using the auxiliary mapping $\overline{\cdot}$ and the special name "$\bullet$". For example "$(\!|\mathsf{in}\ (\mathsf{out}\ \mathsf{a})|\!) = \mathsf{in}\ \bullet$". Recall that in Meta✶ "$x\mathtt{[}P\mathtt{]}$" is an abbreviation for "$x\mathtt{[]}.P$", and that "$_*$" linearizes composed messages (like $(\mathsf{a.b})_*P) = \mathsf{a.b}.P$). The encoding erases type annotations; this is okay because TMA's rewriting rules only copy type annotations around without any other effect. The type embedding in Sec. 3.3 will recover type information by different means. Thm. 3.3 expresses correctness of the encoding and the set $\mathcal{A}$. The correspondence is modulo structural equivalence because of some differences in structural equivalences of both systems. The second implication means that range of the encoding is closed under rewriting.

THEOREM 3.3. *It holds that*

$$\begin{array}{lll} B_0 \to B_1 & \textit{implies} & \exists B_0', B_1' : B_0 \equiv B_0' \ \&\ (\!|B_0'|\!) \overset{\mathcal{A}}{\hookrightarrow} (\!|B_1'|\!) \ \&\ B_1' \equiv B_1. \\ (\!|B_0|\!) \overset{\mathcal{A}}{\hookrightarrow} P_1 & \textit{implies} & \exists B_1 : (\!|B_1|\!) \equiv P_1 \ \&\ B_0 \to B_1. \end{array}$$

$F \in \mathsf{BasicEnv} = \mathsf{BasicName} \xrightarrow{\text{fin}} \mathsf{AMessageType}$
$I \in \mathsf{TypeInfo} = \mathsf{BasicEnv} \times \mathsf{BasicEnv} \times \mathsf{AExchangeType}$

abbreviations:
$(I_{\mathsf{fnb}}, I_{\mathsf{ib}}, I_{\mathsf{top}}) = I$
$I_{\mathsf{env}} = I_{\mathsf{fnb}} \cup I_{\mathsf{ib}}$

$\mathsf{benv}_E(a) = W$ iff $E(a^i) = W$ for some i
$\mathsf{cenv}_B(a) = W$ iff B has a subprocess $(\ldots, a^i : W, \ldots).B_0$
$\mathsf{nenv}_B(a) = W$ iff $W = \mathsf{Amb}[T]$ and B has a subprocess $(\nu a^i : W)B_0$

Figure 11: Functions and structures used to extract type information necessary for the construction of a Poly✶ type.

3.3 Embedding of TMA in Poly✶

Ideally, we could translate a TMA typing pair (E, T) into a Poly✶ type π with corresponding meaning. Because Poly✶ types describe Meta✶ process structures including positions and shapes of input binders, it is not possible to translate (E, T) alone into a type π because (E, T) does not say what input binders are allowed. It is enough to know the set of input-bound names and their types, because this determines the shapes of input binders and their allowed positions. A similar situation occurs with ν-bound names because, again, to properly describe the structure of a process in Poly✶ we need to know allowed types and positions of private names.

Although we could not provide a one-to-one correspondence between TMA typings and Poly✶ types, we can exactly emulate the TMA typing relation using Poly✶'s. From the triple (E, T, B) we construct the Poly✶ type π such that $E \vdash B : T$ holds iff $\vdash (\!|B|\!) : \pi$ holds in Poly✶. Furthermore, the construction of π depends on only type information in B and not any other part of B's structure.

Fig. 11 defines notions used to collect necessary information. A basic environment F assigns TMA message types to basic names; it is just like a TMA environment except for basic names instead of names. We use basic environments to store types of free and bound names. The well-scopedness rules S3 and S4 ensure there is no ambiguity in using only basic names to refer to typed names in a process and an environment.

The type information I encapsulates what is needed to construct a Poly✶ type. It is a triple of two basic environments and one exchange type. The first basic environment of I, denoted I_{fnb}, stores types of free and ν-bound names. We use just one basic environment to store types of both free and ν-bound names to show the construction need not distinguish between them. In contrast, the construction needs to know which basic names are input-bound in a process. Thus input-bound names with their types are stored by the second basic environment of I, denoted I_{ib}. The third member of I, denoted I_{top}, is the top-level exchange type (i.e., the T from $E \vdash B : T$ we are translating). The abbreviation I_{env} denotes $I_{\mathsf{fnb}} \cup I_{\mathsf{ib}}$. The well-scopedness rule S1 ensures I_{env} is a function.

The functions benv, cenv, and nenv extract basic names and their types from a process and an environment. The basic environment benv_E is the same as E except that it forgets upper indexes of names. The basic environment cenv_B describes input-bound names from B. Finally nenv_B describes ν-bound names

Set of nodes of a shape graph (and correspondence functions):

$$\mathsf{types}_I = \{I_{\mathsf{top}}\} \cup \{T\colon \mathsf{Amb}[T] \in \mathsf{rng}(I_{\mathsf{env}})\} \qquad \mathsf{nodeof}_I = \mathsf{typeof}_I^{-1}$$

Let nodes_I be an arbitrary but fixed set of nodes such that there exist the bijection typeof_I from nodes_I into types_I.

Action types describing legal capabilities:

$$\mathsf{namesof}_I(W) = \{a\colon I_{\mathsf{env}}(a) = W\}$$
$$\mathsf{allowedin}_I(T) = \mathsf{moves}_I \cup \mathsf{opens}_I(T) \cup \mathsf{comms}_I(T)$$
$$\mathsf{moves}_I = \{\mathsf{in}\ a, \mathsf{out}\ a\colon \exists T.\, a \in \mathsf{namesof}_I(\mathsf{Amb}[T])\}$$
$$\mathsf{opens}_I(T) = \{\mathsf{open}\ a\colon a \in \mathsf{namesof}_I(\mathsf{Amb}[T])\} \cup \mathsf{namesof}_I(\mathsf{Cap}[T])$$
$$\mathsf{msgs}_I(\mathsf{Amb}[T]) = \mathsf{namesof}_I(\mathsf{Amb}[T])$$
$$\mathsf{msgs}_I(\mathsf{Cap}[T]) = \mathsf{namesof}_I(\mathsf{Cap}[T]) \cup \{(\mathsf{moves}_I \cup \mathsf{opens}_I(T))*\}$$
$$\mathsf{comms}_I(\mathsf{Shh}) = \emptyset$$
$$\mathsf{comms}_I(W_1 \otimes \cdots \otimes W_k) = \{\texttt{<}\mu_1, \ldots, \mu_k\texttt{>}\colon \mu_i \in \mathsf{msgs}_I(W_i)\} \cup \{(a_1, \ldots, a_k)\colon I_{\mathsf{ib}}(a_i) = W_i \ \&\ (i \neq j \Rightarrow a_i \neq a_j)\}$$

Construction of shape predicates and embedding of type judgments:

$$\langle\!|I|\!\rangle = \{\chi \xrightarrow{\alpha} \chi\colon \alpha \in \mathsf{allowedin}_I(\mathsf{typeof}_I(\chi)) \ \&\ \chi \in \mathsf{nodes}_I\} \cup \{\chi \xrightarrow{a\,[]} \chi'\colon a \in \mathsf{namesof}_I(\mathsf{Amb}[\mathsf{typeof}_I(\chi')]) \ \&\ \chi, \chi' \in \mathsf{nodes}_I\}$$
$$[\![I]\!] = \langle \langle\!|I|\!\rangle, \mathsf{nodeof}_I(I_{\mathsf{top}})\rangle \qquad \mathsf{typeinfo}(E, B, T) = (\mathsf{benv}_E \cup \mathsf{nenv}_B, \mathsf{cenv}_B, T)$$

Figure 12: Construction of Poly★ type embedding.

of B which have some ambient type. We need only ambient types because only they can be types of ν-bound names in typable TMA processes; this is where our translation enforces thisTMA property. Altogether, the type information extracted from (E, B, T) is $(\mathsf{benv}_E \cup \mathsf{nenv}_B, \mathsf{cenv}_B, T)$.

EXAMPLE 3.4. *The type information $I = (I_{\mathsf{fnb}}, I_{\mathsf{ib}}, I_{\mathsf{top}})$ for our running example is:*

$$I_{\mathsf{fnb}} = \{\mathsf{d} \mapsto \mathsf{Amb}[\mathbf{1}], \mathsf{p} \mapsto \mathsf{Amb}[\mathbf{1}]\} \quad I_{\mathsf{ib}} = \{\mathsf{x} \mapsto \mathsf{Cap}[\mathbf{1}]\} \quad I_{\mathsf{top}} = \mathsf{Cap}[\mathbf{1}]$$

The main idea of the construction is as follows. The constructed shape graph contains exactly one node for every exchange type of some ambient location. It means one node for the top-level type T, and one node for T' whenever some basic name contained in I has the type $\mathsf{Amb}[T']$. The node corresponding to the top-level type becomes the shape predicate root. Each node corresponding to some T has self-loops which describe all capabilities and communication actions which a process of the type T can execute. For example, when $I_{\mathsf{fnb}}(\mathsf{a}) = \mathsf{Amb}[\mathbf{1}]$ then every node would have a self-loop labeled by "in a" because in-capabilities can be executed by any process. On the other hand only the node which corresponds to **1** would allow "open a" because only processes of the type **1** can legally execute it. Finally, following an edge labeled with "a[]" means entering a. Thus the edge has lead to the node that corresponds to the a's type. In the above example, the shape graph would contain edges labeled with the ambient action type "a[]" from any node to the node corresponding to **1**.

The construction starts by building the node set of a shape predicate (Fig. 12). All the exchange types of ambient locations are gathered in the set types_I. These types are put in bijective correspondence (via the two mutually inverse bijections nodeof_I and typeof_I) with the members of nodes_I.

EXAMPLE 3.5. *In the running example we have* $\mathsf{types}_I = \{\mathsf{Cap}[\mathbf{1}], \mathbf{1}\}$. *Let us choose* $\mathsf{nodes}_I = \{\mathtt{R}, 1\}$ *and define the bijections such that* $\mathsf{nodeof}_I(\mathbf{1}) = 1$.

The middle part of Fig. 12 defines two important auxiliary functions used by the translation: $\mathsf{namesof}_I$ and $\mathsf{allowedin}_I$. The first one provides for each message type W the set $\mathsf{namesof}_I(W)$ of all basic names declared with the type W by I. The second one provides for the TMA exchange type T the set of Poly★ action types $\mathsf{allowedin}_I(T)$ which describe (translations of) all capabilities and action prefixes which are allowed to be legally executed by a process of the type T. The set $\mathsf{allowedin}_I(T)$ consists of three parts: moves_I, $\mathsf{opens}_I(T)$, and $\mathsf{comms}_I(T)$. The action types in moves_I describe all in/out capabilities which can be constructed from ambient basic names in I. In other words all in/out capabilities which can be legally executed by a process of some appropriate type from I. The set does not depend on T because in/out capabilities can be executed by any process. The set $\mathsf{opens}_I(T)$ describe capabilities which can be executed by a process of the type T. It consists of open-capabilities constructible from ambient names in I and from those basic names which are directly declared to the type $\mathsf{Cap}[T]$ by I. The second part of $\mathsf{opens}_I(T)$ describes names which are supposed to be instantiated by communication to some executable capabilities. The set $\mathsf{comms}_I(T)$ describes communication actions which can be executed by a process of type T. Its first part describes output- and the second one input-communication actions. The first part is defined using the auxiliary function msgs_I which for each TMA message type W defines the set of Poly★ message types describing all messages of the type W constructible from names in I.

EXAMPLE 3.6. *Relevant sets for our example are:*

$\mathsf{namesof}_I(\mathsf{Amb}[\mathbf{1}]) = \{\mathsf{d}, \mathsf{p}\}$ $\quad$ $\mathsf{opens}_I(\mathbf{1}) = \{\mathsf{open\ d}, \mathsf{open\ p}, \mathsf{x}\}$

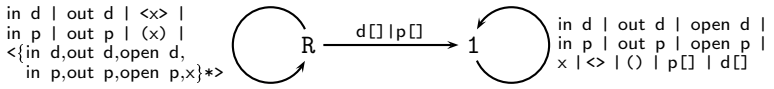
$\mathsf{namesof}_I(\mathsf{Cap}[\mathbf{1}]) = \{\mathsf{x}\}$ $\quad$ $\mathsf{opens}_I(\mathsf{Cap}[\mathbf{1}]) = \emptyset$

$\mathsf{comms}_I(\mathbf{1}) = \{\mathsf{<>}, \mathsf{()}\}$ $\quad$ $\mathsf{moves}_I = \{\mathsf{in\ d}, \mathsf{in\ p}, \mathsf{out\ d}, \mathsf{out\ p}\}$

$\mathsf{comms}_I(\mathsf{Cap}[\mathbf{1}]) = \{\mathsf{<x>}, \mathsf{<\{in\ d, in\ p, out\ d, out\ p, open\ d, open\ p, x\}*>}, \mathsf{(x)}\}$

Finally the third part of Fig. 12 describes the construction of the shape graph $\langle\!|I|\!\rangle$ and the shape predicate $\{\!|I|\!\}$ from I. The first part of the definition of $\langle\!|I|\!\rangle$ describes self-loops of nodes. These self loops describe actions allowed to be executed by the node's corresponding type. Edges from the second part of $\langle\!|I|\!\rangle$ describe transitions among nodes. Any edge labeled by the ambient action type "$a[\,]$" always leads to the node which corresponds to the exchange type allowed inside a. Finally Fig. 12 shows how type information is extracted from (E, B, T).

EXAMPLE 3.7. *The resulting shape predicate* $\{\!|I|\!\} = \langle G, \mathtt{R}\rangle$ *in our example has the root* R *and its shape graph* G *is below. We merge labels of edges with the same source and destination into one using "|".*



Correctness of the translation is expressed by Thm. 3.8. The condition on $\mathsf{fn}(B)$ is necessary for Poly★ shape predicates handle all names sharing the basic part uniformly, and it is naturally satisfied for all desirable applications.

THEOREM 3.8. *If* $\mathsf{fn}(B) \subseteq \mathsf{dom}(E)$ *then* $E \vdash B : T \Leftrightarrow \vdash (\!|B|\!) : (\!|\mathsf{typeinfo}(E, B, T)|\!)$.

THEOREM 3.9. $(\!|\mathsf{typeinfo}(E, B, T)|\!)$ *is a* Poly★ $\mathcal{A}$*-type (up to the presence of flow edges).*

3.4 Further Possibilities of Poly★ Types for MA

The above construction of the Poly★ type can be altered in many ways to improve expressiveness. In subsequent work [CGG99], Cardelli, Ghelli, and Gordon define a type system which can ensure that some ambients stay immobile or that their boundaries are never dissolved. We can easily adapt our construction to reflect these needs by removing appropriate capabilities from self loops of nodes. In our construction we have one node for each type of some ambient location. Alternatively we can have a separate node for each ambient. This allows us to express more refine properties of ambients, and we can express predicates defined by another subsequent work of the above authors concerning ambient groups [CGG00].

We are not forced to use only self loops to describe actions inside ambients. We can use an edge sequence to describe action execution order similarly to ";" sequencing of session types [Hon93, THK94, HVK98, HYC08]. For example, we can express that the action "out a" is followed by "in b". Moreover, we can take an advantage of Poly★'s spatial polymorphism (see [MW05]) to express location-dependent properties of ambients, e.g., that the ambient a can be opened only inside the ambient b.

4 Poly★ and Static Analysis of BioAmbients

BioAmbients, introduced by Regev et al. [RPS+04], is a process calculus for modeling biomolecular systems. Nielson et al. [NNPR07] designed a static analysis system for BioAmbients (hereafter SABA) which conservatively over-approximates the states that a system can evolve to. We show that Poly★ can provide the same information as SABA and can do better. Regev et al. present BioAmbients with the choice operator to express computation options and with replication. SABA uses the **rec** operator[1] instead of replication. We name the SABA's version of BioAmbients BA. We define BA$^-$, a choice-free variant of BA

[1] Process calculi with **rec** additionally introduce process variables, say X, and processes of the form **rec** $X.P$ which behave like P with **rec** $X.P$ substituted for X.

Syntax of BA^-*:*

$$
\begin{array}{rrcl}
n, m \in & \mathsf{BioName} & = & \mathsf{Name}\backslash\{\bullet, [], \hat{}, ?, !, \{, \}, \star, \mathsf{local}, \ldots, \mathsf{s2s}, \mathsf{enter}, \ldots, \mathsf{merge}\text{-}\} \\
d \in & \mathsf{BioDirection} & ::= & \mathsf{local} \mid \mathsf{p2c} \mid \mathsf{c2p} \mid \mathsf{s2s} \\
N \in & \mathsf{BioCapability} & ::= & \mathsf{enter}\ n \mid \mathsf{accept}\ n \mid \mathsf{exit}\ n \mid \mathsf{expel}\ n \mid \mathsf{merge}\text{+}\ n \mid \mathsf{merge}\text{-}\ n \\
B \in & \mathsf{BioProcess} & ::= & 0 \mid B_0 \mid B_1 \mid [B]^a \mid N.B \mid d\ n?\{m\}.B \mid d\ n!\{m\}.B \mid\ !B \mid (\nu n)B
\end{array}
$$

Structural equivalence of BA^- *is generated by:*

$$
\begin{array}{ll}
B \mid 0 \equiv B & (\nu n)0 \equiv 0 \\
B_0 \mid B_1 \equiv B_1 \mid B_0 & (\nu n)([B]^a) \equiv [(\nu n)B]^a \\
B_0 \mid (B_1 \mid B_2) \equiv (B_0 \mid B_1) \mid B_2 & B_0 \mid (\nu n)B_1 \equiv (\nu n)(B_0 \mid B_1) \text{ if } n \notin \mathsf{fn}(B_0) \\
!0 \equiv 0 & (\nu n)(\nu m)B \equiv (\nu m)(\nu n)B \\
!B \equiv B \mid !B &
\end{array}
$$

Rewriting relation of BA^-*:*

$$
\begin{array}{rcl}
[\mathsf{enter}\ n.B_0 \mid B_1]^a \mid [\mathsf{accept}\ n.B_2 \mid B_3]^b & \to & [[B_0 \mid B_1]^a \mid B_2 \mid B_3]^b \\
[[\mathsf{exit}\ n.B_0 \mid B_1]^a \mid \mathsf{expel}\ n.B_2 \mid B_3]^b & \to & [B_0 \mid B_1]^a \mid [B_2 \mid B_3]^b \\
[\mathsf{merge}\text{+}\ n.B_0 \mid B_1]^a \mid [\mathsf{merge}\text{-}\ n.B_2 \mid B_3]^b & \to & [B_0 \mid B_1 \mid B_2 \mid B_3]^a \\
\mathsf{local}\ n?\{m_0\}.B_0 \mid \mathsf{local}\ n!\{m_1\}.B_1 & \to & B_0\{m_0 \mapsto m_1\} \mid B_1 \\
\mathsf{p2c}\ n?\{m_0\}.B_0 \mid [\mathsf{c2p}\ n!\{m_1\}.B_1 \mid B_2]^a & \to & B_0\{m_0 \mapsto m_1\} \mid [B_1 \mid B_2]^a \\
[\mathsf{c2p}\ n?\{m_0\}.B_0 \mid B_1]^a \mid \mathsf{p2c}\ n!\{m_1\}.B_2 & \to & [B_0\{m_0 \mapsto m_1\} \mid B_1]^a \mid B_2 \\
[\mathsf{s2s}\ n?\{m_0\}.B_0 \mid B_1]^a \mid [\mathsf{s2s}\ n!\{m_1\}.B_2 \mid B_3]^b & \to & [B_0\{m_0 \mapsto m_1\} \mid B_1]^a \mid [B_2 \mid B_3]^b
\end{array}
$$

$$
\frac{B_0 \to B_1}{(\nu n)B_0 \to (\nu n)B_1} \qquad \frac{B_0 \to B_1}{[B_0]^a \to [B_1]^a} \qquad \frac{B_0 \to B_1}{B_0 \mid B_2 \to B_1 \mid B_2}
$$

$$
\frac{B_0' \equiv B_0 \quad B_0 \to B_1 \quad B_1 \equiv B_1'}{B_0' \to B_1'}
$$

Figure 13: Syntax and semantics of BA^-.

with replication, and compare information provided by Poly★ types with those computed by SABA. Finally we show how to adapt our method to handle choice as well.

4.1 Brief Description of BioAmbients

BA differs from MA in several ways. Ambients are anonymous, that is, are not labeled with names. It implies that capabilities can no longer use names to refer to ambients. Thus capabilities come in require/allow pairs synchronized by names, for example, "enter a/accept a". Then an appropriate action is performed when two ambients containing corresponding parts are found in a required position. The open capability is replaced by an operation that merges two sibling ambients. Communication is channel-based, that is, both a sender and receiver have to agree on a channel name for communication to happen. Moreover, communication is allowed also across some ambient boundaries, and only single names are exchanged.

Fig. 13 gives the syntax of BA^-. As for TMA, we build processes from Meta★

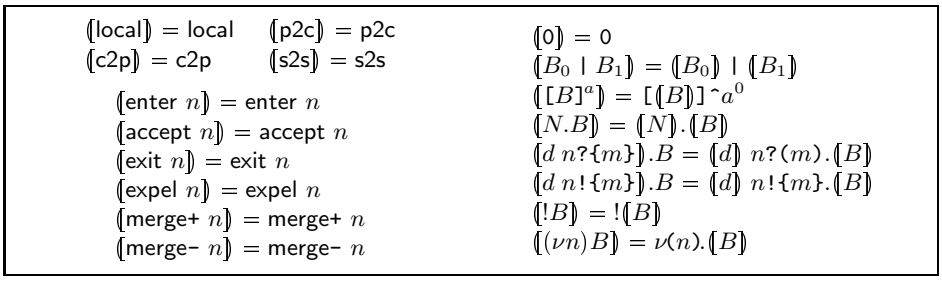


Figure 14: Encoding of BA⁻ processes in Meta✶.

names to ease comparison. Some names are reserved for translating BA⁻ processes into Meta✶. The capability "enter n" instructs an ambient to enter a sibling containing a corresponding "accept n". "exit n" instructs an ambient to exit its parent ambient provided it allows it with the "expel n" capability. Finally, "merge+ n" instructs an ambient to merge with a sibling containing "merge− n". Communication is in four directions: between processes in the same ambient (local), between processes in sibling ambients (s2s), from a parent ambient to its child (p2c), and from a child to the parent (c2p). Communication output is "$d\ n\,!\{m\}$" where n is the channel name, d is the desired direction, and m is the name being sent. The input prefix "$d\ n\,?\{m\}$" (input-)binds the name m.

Static analysis must refer to ambients to track changes, so SABA introduces ambient labels with no influence on the semantics. We translate these labels as Meta✶ basic names and write $[B]^a$ for an ambient labeled a. We identify α-convertible processes. We require all processes to satisfy well-scopedness rules S1-2 from Sec. 3.1, and that basic names used as ambient labels are not used as ordinary names. The semantics of BA⁻ is in Fig. 13. Structural equivalence $\equiv$ is the smallest binary relation on BA⁻ processes that satisfies the rules in the middle part of Fig. 13 and is congruent with process constructors.

4.2 Instantiation of Meta✶ to BA⁻

As in the case of TMA we define the encoding $(\!|\cdot|\!)$ of BA⁻ processes in Meta✶, and the set $\mathcal{B}$ that describes rewriting rules of BA⁻. The encoding is defined in Fig.14 and the instantiation $\mathcal{B}$ of Meta✶ to BA⁻ is given as follows:

$\mathcal{B} = \{$ **active**{ $\mathring{P}$ **in** [$\mathring{P}$]^$\mathring{a}$ },
rewrite{ [enter $\mathring{n}$.$\mathring{P}$ | $\mathring{Q}$]^$\mathring{a}$ | [accept $\mathring{n}$.$\mathring{R}$ | $\mathring{S}$]^$\mathring{b}$ ↪ [[$\mathring{P}$ | $\mathring{Q}$]^$\mathring{a}$ | $\mathring{R}$ | $\mathring{S}$]^$\mathring{b}$ },
rewrite{ [[exit $\mathring{n}$.$\mathring{P}$ | $\mathring{Q}$]^$\mathring{a}$ | expel $\mathring{n}$.$\mathring{R}$ | $\mathring{S}$]^$\mathring{b}$ ↪ [$\mathring{P}$ | $\mathring{Q}$]^$\mathring{a}$ | [$\mathring{R}$ | $\mathring{S}$]^$\mathring{b}$ },
rewrite{ [merge+ $\mathring{n}$.$\mathring{P}$ | $\mathring{Q}$]^$\mathring{a}$ | [merge− $\mathring{n}$.$\mathring{R}$ | $\mathring{S}$]^$\mathring{b}$ ↪ [$\mathring{P}$ | $\mathring{Q}$ | $\mathring{R}$ | $\mathring{S}$]^$\mathring{a}$ },
rewrite{ local $\mathring{n}$?{$\mathring{x}$}.$\mathring{P}$ | local $\mathring{n}$!{$\mathring{m}$}.$\mathring{Q}$ ↪ {$\mathring{x}$:= $\mathring{m}$}$\mathring{P}$ | $\mathring{Q}$ },
rewrite{ p2c $\mathring{n}$?{$\mathring{x}$}.$\mathring{P}$ | [c2p $\mathring{n}$!{$\mathring{m}$}.$\mathring{Q}$ | $\mathring{R}$]^$\mathring{a}$ ↪ {$\mathring{x}$:= $\mathring{m}$}$\mathring{P}$ | [$\mathring{Q}$ | $\mathring{R}$]^$\mathring{a}$ },
rewrite{ [c2p $\mathring{n}$?{$\mathring{x}$}.$\mathring{P}$ | $\mathring{Q}$]^$\mathring{a}$ | p2c $\mathring{n}$!{$\mathring{m}$}.$\mathring{R}$ ↪ [{$\mathring{x}$:= $\mathring{m}$}$\mathring{P}$ | $\mathring{Q}$]^$\mathring{a}$ | $\mathring{R}$ },
rewrite{ [s2s $\mathring{n}$?{$\mathring{x}$}.$\mathring{P}$ | $\mathring{Q}$]^$\mathring{a}$ | [s2s $\mathring{n}$!{$\mathring{m}$}.$\mathring{R}$ | $\mathring{S}$]^$\mathring{b}$ ↪
[{$\mathring{x}$:= $\mathring{m}$}$\mathring{P}$ | $\mathring{Q}$]^$\mathring{a}$ | [$\mathring{R}$ | $\mathring{S}$]^$\mathring{b}$ } $\}$

Analogously to the case of TMA, the correctness of the encoding and of the rewriting rules is expressed by the following theorem.

THEOREM 4.1. *It holds that*

$$B_0 \to B_1 \quad \textit{implies} \quad \exists B_0', B_1' : B_0 \equiv B_0' \;\&\; (\!|B_0'|\!) \overset{\mathcal{B}}{\hookrightarrow} (\!|B_1'|\!) \;\&\; B_1' \equiv B_1.$$
$$(\!|B_0|\!) \overset{\mathcal{B}}{\hookrightarrow} P_1 \quad \textit{implies} \quad \exists B_1 : (\!|B_1|\!) \equiv P_1 \;\&\; B_0 \to B_1.$$

4.3 Differences of SABA and the Original Analysis

There are some technical differences between the version of SABA presented in the original work of Nielson et al. [NNPR07] and the version of SABA presented in this paper. However, this paper defines its slightly different version just in order to simplify presentation of formal definitions and results, and both versions can be considered equivalent with respect to results they compute for equivalent processes. That is why we do not introduce different names for the two systems.

The first and the main difference is that SABA in this version work with BA$^-$ and the original one with BA. The differences between BA$^-$ and BA are as follows:

(1) BA$^-$ restricts use of the **rec** operator just to the cases that can be equivalently expressed by replication
(2) BA$^-$ does not allow the use of the choice operator
(3) BA$^-$ processes are identified up to the α-equivalence while BA processes are not
(4) BA$^-$ processes are build from Meta$\star$ names while BA processes are not
(5) BA$^-$ uses Meta$\star$ basic names to represent ambient labels while BA use a separate set of labels

The reason for restrictions (1-2) is that Meta$\star$ does not handle appropriate operators. Differences (3-5) are purely technical matters destined to simplify the presentation.

The second difference between the two analysis is related to α-renaming. The original system supposes that the canonical name $\lfloor n \rfloor$ is assigned to each name n, and that canonical names are preserved under α-renaming. The original system assumes that some arbitrary but fixed assignment with the above property exist but no further details are specified. SABA in this paper defines canonical names using Meta$\star$ basic names as $\lfloor a^i \rfloor = a$, and uses the property of Meta$\star$ that basic names are preserved under α-renaming.

The third difference is in the representation of results of the analysis. In order to simplify the presentation, SABA in this paper uses Meta$\star$ and Poly$\star$ entities to represent the results while the original system uses the following sets. The set **Ambient** is the set of ambient labels, the set **Name** is the set of canonical names, and the set **Cap** is the set of canonical capabilities and communication prefixes build from canonical names instead of names. Then a result of the original analysis has the form $(\mathcal{I}, \mathcal{R})$ where:

$$\mathcal{I} \subseteq \mathbf{Ambient} \times (\mathbf{Ambient} \cup \mathbf{Cap}) \qquad \mathcal{R} \subseteq \mathbf{Name} \times \mathbf{Name}$$

We use disjoint subsets of basic names for both ambient labels **Ambient** and canonical names **Name**, and thus canonical capabilities are already contained in ActionType. Moreover in this paper we write $\mathcal{I}(a, \texttt{[]\^{}}b)$ instead of $\mathcal{I}(a, b)$ again for the reasons of simplification. Thus $\mathcal{I}$ in SABA of this paper becomes a subset of BasicName $\times$ ActionType and $\mathcal{R}$ becomes a subset of BasicName $\times$ BasicName.

The second and the third difference are purely technical. But the first difference restrict the set of processes on which the analysis can be executed. The original analysis can be executed for all BA^- processes because replication can be expressed by the **rec** operator defining "$!B$" as "**rec** $X.(B \mid X)$". While the original analysis can be executed on additional processes, both analysis yield equivalent results for BA^- processes. Thus both analysis can be considered equivalent on these processes.

4.4 Brief Description of SABA

SABA takes a BA^- process as an input and its output collects information about possible contents of ambients in any process that the input process can evolve to. A result of SABA analysis is the pair $(\mathcal{I}, \mathcal{R})$ where

$$\begin{aligned} \mathcal{I} \in \mathsf{SABAInsides} &= \mathcal{P}_{\mathrm{fin}}(\mathsf{BasicName} \times \mathsf{FormType}) \\ \mathcal{R} \in \mathsf{SABARenamings} &= \mathcal{P}_{\mathrm{fin}}(\mathsf{BasicName} \times \mathsf{BasicName}) \end{aligned}$$

For every ambient, $\mathcal{I}$ collects information about possible child ambients, capabilities, and communication prefixes contained in it. For example $(\mathsf{a}, \texttt{[]\^{}}\mathsf{b}) \in \mathcal{I}$ says[2] that the ambient (with the label) a can have a child ambient b, while $(\mathsf{a}, \mathsf{enter\ n}) \in \mathcal{I}$ says that an ambient with the label a can possibly contain (and execute) the capability "$\mathsf{enter\ n}^i$" for any i. Input-bound names are handled in a special way. When a capability was built from an input-bound name then it would not be contained in $\mathcal{I}$. Instead, $\mathcal{I}$ contains all its actual instantiations introduced by communication. For example, for the input process "$\mathsf{local\ a?\{x\}.enter\ x.0 \mid local\ a!\{b\}.0}$", the $\mathcal{I}$ part of the result contains "$\mathsf{enter\ b}$" but not "$\mathsf{enter\ x}$". The set $\mathcal{R}$ describes possible name renamings invoked by communication. For example $(\mathsf{n}, \mathsf{m}) \in \mathcal{R}$ says that communication can rename n to m.

SABA defines the predicate $(\mathcal{I}, \mathcal{R}) \models^l B$ meaning that B matches the structure allowed by $(\mathcal{I}, \mathcal{R})$ inside the ambient l. The name "$\star$" is used to refer to the top level location. The definition is presented in Fig. 15. Closure conditions which reflect BA rewriting rules are presented in Fig. 16. Every valid SABA result has to satisfy these conditions. The result of SABA for the input B is the smallest pair $(\mathcal{I}, \mathcal{R})$ such that (1) it holds that $(\mathcal{I}, \mathcal{R}) \models^\star B$, (2) it satisfies all closure conditions from Fig. 15, and (3) for every free name $a^i \in \mathsf{fn}(B)$ it holds that $(a, a) \in \mathcal{R}$. SABA grants that the structure described by a valid result is closed under rewritings.

[2] In the original paper [NNPR07] the set $\mathcal{I}$ contains (a, b) instead of $(a, \texttt{[]\^{}}b)$. This technical difference allows easier formulation of results.

$$
\begin{array}{lll}
(\mathcal{I},\mathcal{R}) \models^l \mathsf{0} & \text{iff} & \text{true} \\
(\mathcal{I},\mathcal{R}) \models^l B_0 \mid B_1 & \text{iff} & (\mathcal{I},\mathcal{R}) \models^l B_0 \;\&\; (\mathcal{I},\mathcal{R}) \models^l B_1 \\
(\mathcal{I},\mathcal{R}) \models^l [B]^{l_0} & \text{iff} & \mathcal{I}(l, \texttt{[]\^{}}l_0) \;\&\; (\mathcal{I},\mathcal{R}) \models^{l_0} B \\
(\mathcal{I},\mathcal{R}) \models^l N.B & \text{iff} & (\mathcal{I},\mathcal{R}) \models^l N \;\&\; (\mathcal{I},\mathcal{R}) \models^l B \\
(\mathcal{I},\mathcal{R}) \models^l d\ n?\{m\}.B & \text{iff} & (\mathcal{I},\mathcal{R}) \models^l d\ n?\{m\} \;\&\; (\mathcal{I},\mathcal{R}) \models^l B \\
(\mathcal{I},\mathcal{R}) \models^l d\ n!\{m\}.B & \text{iff} & (\mathcal{I},\mathcal{R}) \models^l d\ n!\{m\} \;\&\; (\mathcal{I},\mathcal{R}) \models^l B \\
(\mathcal{I},\mathcal{R}) \models^l !B & \text{iff} & (\mathcal{I},\mathcal{R}) \models^l B \\
(\mathcal{I},\mathcal{R}) \models^l (\nu a^i)B & \text{iff} & \mathcal{R}(a,a) \;\&\; (\mathcal{I},\mathcal{R}) \models^l B
\end{array}
$$

$$
\begin{array}{lll}
(\mathcal{I},\mathcal{R}) \models^l \mathsf{enter}\ a^i & \text{iff} & \forall b : \mathcal{R}(a,b) \Rightarrow \mathcal{I}(l, \mathsf{enter}\ b) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{accept}\ a^i & \text{iff} & \forall b : \mathcal{R}(a,b) \Rightarrow \mathcal{I}(l, \mathsf{accept}\ b) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{exit}\ a^i & \text{iff} & \forall b : \mathcal{R}(a,b) \Rightarrow \mathcal{I}(l, \mathsf{exit}\ b) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{expel}\ a^i & \text{iff} & \forall b : \mathcal{R}(a,b) \Rightarrow \mathcal{I}(l, \mathsf{expel}\ b) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{merge+}\ a^i & \text{iff} & \forall b : \mathcal{R}(a,b) \Rightarrow \mathcal{I}(l, \mathsf{merge+}\ b) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{merge-}\ a^i & \text{iff} & \forall b : \mathcal{R}(a,b) \Rightarrow \mathcal{I}(l, \mathsf{merge-}\ b) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{local}\ a^i!\{b^j\} & \text{iff} & \forall a', b' : \mathcal{R}(a,a') \;\&\; \mathcal{R}(b,b') \Rightarrow \mathcal{I}(l, \mathsf{local}\ a'!\{b'\}) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{local}\ a^i?\{b^j\} & \text{iff} & \forall a' : \mathcal{R}(a,a') \Rightarrow \mathcal{I}(l, \mathsf{local}\ a'?\{b\}) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{p2c}\ a^i!\{b^j\} & \text{iff} & \forall a', b' : \mathcal{R}(a,a') \;\&\; \mathcal{R}(b,b') \Rightarrow \mathcal{I}(l, \mathsf{p2c}\ a'!\{b'\}) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{p2c}\ a^i?\{b^j\} & \text{iff} & \forall a' : \mathcal{R}(a,a') \Rightarrow \mathcal{I}(l, \mathsf{p2c}\ a'?\{b\}) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{c2p}\ a^i!\{b^j\} & \text{iff} & \forall a', b' : \mathcal{R}(a,a') \;\&\; \mathcal{R}(b,b') \Rightarrow \mathcal{I}(l, \mathsf{c2p}\ a'!\{b'\}) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{c2p}\ a^i?\{b^j\} & \text{iff} & \forall a' : \mathcal{R}(a,a') \Rightarrow \mathcal{I}(l, \mathsf{c2p}\ a'?\{b\}) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{s2s}\ a^i!\{b^j\} & \text{iff} & \forall a', b' : \mathcal{R}(a,a') \;\&\; \mathcal{R}(b,b') \Rightarrow \mathcal{I}(l, \mathsf{s2s}\ a'!\{b'\}) \\
(\mathcal{I},\mathcal{R}) \models^l \mathsf{s2s}\ a^i?\{b^j\} & \text{iff} & \forall a' : \mathcal{R}(a,a') \Rightarrow \mathcal{I}(l, \mathsf{s2s}\ a'?\{b\})
\end{array}
$$

Figure 15: SABA analysis of BA^- processes.

4.5 Comparing Poly★ Types with SABA's Results

Information provided by SABA results are contained in Poly★ principal typings as well. For example when the shape graph contains '$\chi_0 \xrightarrow{\texttt{[]\^{}}a} \chi_1 \xrightarrow{\texttt{[]\^{}}b} \chi_2$' then it means that an ambient a can possible contain some ambient b. The above two edges can be possibly separated by other edges. We use the following two predicates to extract relevant information from the shape predicate $\pi = \langle G, \chi \rangle$. The action type α_k is said to be under the root of π, written $\mathsf{inroot}_\pi(\alpha_k)$,when G contains a path of edges starting at χ labeled with $\alpha_0, \ldots, \alpha_k$ where none of α_i's preceding α_k is of the shape "$\texttt{[]\^{}}b$". The condition on the shape of α_i's expresses that α_k is not inside any ambient. Similarly, the predicate $\mathsf{inamb}_\pi(a, \alpha_k)$ holds when α_k is contained directly inside the ambient a in G. That is when G contains the edge path "$\texttt{[]\^{}}a, \alpha_0, \ldots, \alpha_k$" starting this time at any node. Again none of the α_i's preceding α_k can have the shape "$\texttt{[]\^{}}b$". We write $\mathsf{inamb}_\pi(\star, \alpha)$ for $\mathsf{inroot}_\pi(\alpha)$.

The following predicate is used to recognized non-instantiated capabilities, that is, those that contain basic names which are bound in some other action types of the shape graph. Let $\mathsf{bn}(G)$ be the set of all basic names which appear inside some input element type $(a_1, \ldots, a_k)$ in G. Let us write $\mathsf{instantiated}_\pi(\alpha)$ when α labels some edge in the shape graph G of π and $\mathsf{fn}(\alpha) \cap \mathsf{bn}(G) = \emptyset$. Then a SABA-like result is constructed from a shape predicate as follows:

$\forall l, l_1, l_2, a:$	$\mathcal{I}(l_1, \mathsf{enter}\ a)\ \&\ \mathcal{I}(l, []\hat{}l_1)\ \&\ \mathcal{I}(l_2, \mathsf{accept}\ a)\ \&\ \mathcal{I}(l, []\hat{}l_2)$ $\Rightarrow \mathcal{I}(l_2, l_1)$
$\forall l, l_1, l_2, a:$	$\mathcal{I}(l_2, \mathsf{exit}\ a)\ \&\ \mathcal{I}(l_1, []\hat{}l_2)\ \&\ \mathcal{I}(l_1, \mathsf{expel}\ a)\ \&\ \mathcal{I}(l, []\hat{}l_1)$ $\Rightarrow \mathcal{I}(l, l_2)$
$\forall l, l_1, l_2, a:$	$\mathcal{I}(l_1, \mathsf{merge+}\ a)\ \&\ \mathcal{I}(l, []\hat{}l_1)\ \&\ \mathcal{I}(l_2, \mathsf{merge-}\ a)\ \&\ \mathcal{I}(l, []\hat{}l_2)$ $\Rightarrow (\forall \alpha : \mathcal{I}(l_2, \alpha) \Rightarrow \mathcal{I}(l_1, \alpha))$
$\forall l, a, b, b':$	$\mathcal{I}(l, \mathsf{local}\ a?\{b\})\ \&\ \mathcal{I}(l, \mathsf{local}\ a!\{b'\})$ $\Rightarrow \mathcal{R}(b, b')$
$\forall l, l_0, a, b, b':$	$\mathcal{I}(l, \mathsf{p2c}\ a!\{b'\})\ \&\ \mathcal{I}(l, []\hat{}l_0)\ \&\ \mathcal{I}(l_0, \mathsf{c2p}\ a?\{b\})$ $\Rightarrow \mathcal{R}(b, b')$
$\forall l, l_0, a, b, b':$	$\mathcal{I}(l, \mathsf{p2c}\ a?\{b\})\ \&\ \mathcal{I}(l, []\hat{}l_0)\ \&\ \mathcal{I}(l_0, \mathsf{c2p}\ a!\{b'\})$ $\Rightarrow \mathcal{R}(b, b')$
$\forall l, l_0, l_1, a, b, b':$	$\mathcal{I}(l_0, \mathsf{s2s}\ a?\{b\})\ \&\ \mathcal{I}(l, []\hat{}l_0)\ \&\ \mathcal{I}(l_1, \mathsf{s2s}\ a!\{b'\})\ \&\ \mathcal{I}(l, []\hat{}l_1)$ $\Rightarrow \mathcal{R}(b, b')$

Figure 16: Closure conditions valid for SABA results.

$\mathcal{I}_\pi = \{(a,\alpha) : \mathsf{inamb}_\pi(a,\alpha)\ \&\ \mathsf{instantiated}_\pi(\alpha)\}$
$\mathcal{R}_\pi = \{(a,b) : (\chi_0 \xrightarrow{\{a \mapsto b\}} \chi_1) \in G\} \cup \{(a,a) : a \in \mathsf{fn}(\alpha) \cap \mathsf{BioName}\ \&\ \mathsf{instantiated}_\pi(\alpha)\}$

The set $\mathcal{R}_\pi$ is constructed from Poly★ flow-edges (described in Sec. 2.3). Thm. 4.2 describes the relation between native SABA results and those constructed from Poly★: Poly★ principal typings contain the information provided by SABA. When (a, α) is in $\mathcal{I}$ but not in $\mathcal{I}_\pi$, then subject reduction of Poly★ ensures the situation predicted by SABA can never happen, in which case Poly★ is more precise.

THEOREM 4.2. *Let B be a* BA$^-$ *process, let $(\mathcal{I}, \mathcal{R})$ be the result of* SABA *analysis of B, and let π be the* Poly★ *principal typing of $(\!|B|\!)$. Then $\mathcal{I}_\pi \subseteq \mathcal{I}$ and $\mathcal{R}_\pi \subseteq \mathcal{R}$.*

BA contains the choice operator ("+") used to express computation options. The process "$B_0 + B_1$" behaves like B_0 or B_1 but only one of them. For the reasons of static analysis we can over approximate "$B_0 + B_1$" with "$B_0 \mid B_1$". This is how SABA handles choice. When we extend BA$^-$ with choice and translate "$(\!|B_0 + B_1|\!)$" to Meta★ as "$(\!|B_0|\!) \mid (\!|B_1|\!)$" then the result stated by Thm. 4.2 remains valid even with the Meta★ instantiation given by $\mathcal{B}$ that does not handle choice. A proper handling of choice in Poly★ is left for further research.

5 Conclusions and Future Work

This paper makes these contributions. (1) We give Meta★ and Poly★ a new approach to α-conversion and name restriction which better supports comparison with other systems. (2) We show the range of Poly★ by comparing with two very different static analyses: TMA and SABA. (3) For TMA, we show how to translate typing judgements to express the same meaning in Poly★; this

used some type information from the process. (4) For SABA, we show that all information in SABA predicates are in Poly✶ principal typings; this works on processes with limited usage of **rec**. (5) For TMA, we detail the correctness of the Meta✶ simulation of TMA's semantics; details of the similar result for BA$^-$ are omitted.

Future work includes further extensions of Meta✶ and Poly✶ and more comparisons. For extensions, priorities are better handling of choice (e.g., because of its use in biological system modeling), and handling of **rec** because in many calculi **rec** is more expressive than replication and better supports describing some recursive behaviors. For comparison with other systems we would like to compare Poly✶ with (1) a system for the π-calculus, (2) with other systems which like Poly✶ use graphs to represent types [Yos96, Kön99, Gad07], and (3) to study in detail the relationship between Poly✶ types and session types [Hon93, THK94, HVK98, HYC08].

References

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Proc. FoSSaCS '98*, volume 1378 of *LNCS*, pages 140–155. Springer-Verlag, 1998.

[CG99] Luca Cardelli and Andrew D. Gordon. Types for mobile ambients. In *Conf. Rec. POPL '99: 26th ACM Symp. Princ. Program. Lang.*, pages 79–92, 1999.

[CGG99] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Mobility types for mobile ambients. In Jiri Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP'99*, volume 1644 of *LNCS*, pages 230–239. Springer-Verlag, July 1999. Extended version appears as Microsoft Research Technical Report MSR-TR-99-32, 1999.

[CGG00] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Ambient groups and mobility types. In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000), Tohoku University, Sendai, Japan*, volume 1872 of *LNCS*, pages 333–347. Springer-Verlag, August 2000.

[Gad07] Fabio Gadducci. Graph rewriting for the π-calculus. *Math. Structures Comput. Sci.*, 17(3):407–437, 2007.

[Hon93] Kohei Honda. Types for dyadic interaction. In *CONCUR 1993*, volume 715 of *LNCS*, pages 509–523, 1993.

[HVK98] Kohei Honda, Vasco Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *Proc. European Symp. on Programming*, volume 1381 of *LNCS*, pages 122–138. Springer-Verlag, 1998.

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Conf. Rec. POPL '08: 35nd ACM Symp. Princ. Program. Lang.*, pages 273–284, 2008.

[Kön99] Barbara König. Generating type systems for process graphs. In *CONCUR 1999*, volume 1664 of *LNCS*, pages 352–367. Springer-Verlag, 1999.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes. *Inform. & Comput.*, 100(1):1–77, September 1992.

[MW04] Henning Makholm and J. B. Wells. Instant polymorphic type systems for mobile process calculi: Just add reduction rules and close. Technical Report HW-MACS-TR-0022, Heriot-Watt Univ., School of Math. & Comput. Sci., November 2004. A shorter successor is [MW05].

[MW05] Henning Makholm and J. B. Wells. Instant polymorphic type systems for mobile process calculi: Just add reduction rules and close. In *Programming Languages & Systems, 14th European Symp. Programming*, volume 3444 of *LNCS*, pages 389–407. Springer-Verlag, 2005. A more detailed predecessor is [MW04].

[NNPR07] Flemming Nielson, Hanne Riis Nielson, Corrado Priami, and Debora Rosa. Control flow analysis for bioambients. *ENTCS*, 180(3):65–79, 2007. A preliminary version appeared at Bio-CONCUR 2003.

[RPS+04] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. Bioambients: An abstraction for biological compartments. *Theoretical Computer Science*, 325(1):141–167, September 2004.

[THK94] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In *PARLE'94*, volume 817 of *LNCS*, pages 398–413. Springer-Verlag, 1994.

[Wel02] J. B. Wells. The essence of principal typings. In *Proc. 29th Int'l Coll. Automata, Languages, and Programming*, volume 2380 of *LNCS*, pages 913–925. Springer-Verlag, 2002.

[Yos96] Nobuko Yoshida. Graph types for monadic mobile processes. In *Foundations of Software Technology and Theoret. Comput. Sci., 16th Conf.*, volume 1180 of *LNCS*, pages 371–386. Springer-Verlag, 1996.

A Proofs of Main Theorems

This section provides proofs or sketches encapsulating main ideas of proofs of the theorems in the paper.

A.1 The Proof of Thm. 3.3

We use the following names for the rewriting rules of TMA which are assigned to the rules from Fig. 8 in the left-right and top-down order: AIn, AOut, AOpen, ACom, AAmb, ANu, APar, and AStr. Similarly we name the Meta✶ rewriting rules from Fig.4 as follows: RRew, RNu, RStr, RPar, RAct.

Lemma A.1. *It holds that*

$$(\!|B\{n_1 \mapsto N_1, \ldots, n_k \mapsto N_k\}|\!) = (\!|B|\!)\{n_1 \mapsto (\!|N_1|\!), \ldots, n_k \mapsto (\!|N_k|\!)\}$$

The following proposition is the left-to-right implication of Thm. 3.3. Additionally, we let C range over TMA processes AProcess in the proof.

Proposition A.2. *Let $B_0 \to B_1$. Then there exist B_0', B_1' such that*

$$B_0 \equiv B_0' \;\&\; (\!|B_0'|\!) \overset{\cdot\mathcal{A}}{\hookrightarrow} (\!|B_1'|\!) \;\&\; B_1' \equiv B_1$$

PROOF. *By induction on the derivation of* $B_0 \to B_1$. *Let it be derived by*

(AIN): *Then, for some* n, m, C_0, C_1, *and* C_1 *we have* $B_0 = n[\mathsf{in}\ m.C_0 \mid C_1] \mid m[C_1]$ *and* $B_1 = m[n[C_0 \mid C_1] \mid C_1]$. *Take the instantiation* $\rho = \{\mathring{\mathtt{a}} \mapsto n, \mathring{\mathtt{b}} \mapsto m, \mathring{\mathtt{P}} \mapsto (\!|C_0|\!), \mathring{\mathtt{Q}} \mapsto (\!|C_1|\!), \mathring{\mathtt{R}} \mapsto (\!|C_1|\!)\}$. *Now, we know that* $\mathbf{rewrite}\{\, \mathring{\mathtt{a}}[\mathsf{in}\ \mathring{\mathtt{b}}.\mathring{\mathtt{P}} \mid \mathring{\mathtt{Q}}] \mid \mathring{\mathtt{b}}[\mathring{\mathtt{R}}] \hookrightarrow \mathring{\mathtt{b}}[\mathring{\mathtt{a}}[\mathring{\mathtt{P}} \mid \mathring{\mathtt{Q}}] \mid \mathring{\mathtt{R}}] \,\} \in \mathcal{A}$. *Moreover it is easy to see that* $(\!|B_0|\!) = [\![\mathring{\mathtt{a}}[\mathsf{in}\ \mathring{\mathtt{b}}.\mathring{\mathtt{P}} \mid \mathring{\mathtt{Q}}] \mid \mathring{\mathtt{b}}[\mathring{\mathtt{R}}]]\!]_\rho$ *and* $(\!|B_1|\!) = [\![\mathring{\mathtt{b}}[\mathring{\mathtt{a}}[\mathring{\mathtt{P}} \mid \mathring{\mathtt{Q}}] \mid \mathring{\mathtt{R}}]]\!]_\rho$. *Take directly* $B_0' = B_0$ *and* $B_1' = B_1$ *and we have* $(\!|B_0'|\!) \overset{\mathcal{A}}{\hookrightarrow} (\!|B_1'|\!)$ *by* RREW.

(AOUT): *Like in case* AIN.

(AOPEN): *Like in case* AIN.

(ACOM): *Similarly to case* AIN *but with the following changes:*

$$
\begin{array}{l}
B_0 = (n_1{:}W_1, \ldots, n_k{:}W_k).C \mid \langle N_1, \ldots, N_k \rangle \\
B_1 = C\{n_1 \mapsto N_1, \ldots, n_k \mapsto N_k\} \\
\rho = \{\mathring{\mathtt{a}}_1 \mapsto n_1, \ldots, \mathring{\mathtt{a}}_k \mapsto n_k, \mathring{\mathtt{M}}_1 \mapsto (\!|N_1|\!), \ldots, \mathring{\mathtt{M}}_k \mapsto (\!|N_k|\!), \mathring{\mathtt{P}} \mapsto \mathtt{0}, \mathring{\mathtt{Q}} \mapsto (\!|C|\!)\} \\
P_0 = [\![(\mathring{\mathtt{a}}_1, \ldots, \mathring{\mathtt{a}}_k).\mathring{\mathtt{Q}} \mid \langle \mathring{\mathtt{M}}_1, \ldots, \mathring{\mathtt{M}}_k \rangle.\mathring{\mathtt{P}}]\!]_\rho = (n_1, \ldots, n_k).(\!|C|\!) \mid \langle (\!|N_1|\!), \ldots, (\!|N_k|\!) \rangle.\mathtt{0} \\
P_1 = [\![\mathring{\mathtt{P}} \mid \{\mathring{\mathtt{a}}_1 := \mathring{\mathtt{M}}_1, \ldots, \mathring{\mathtt{a}}_k := \mathring{\mathtt{M}}_k\}\mathring{\mathtt{Q}}]\!]_\rho = \mathtt{0} \mid (\!|C|\!)\{n_1 \mapsto (\!|N_1|\!), \ldots, n_k \mapsto (\!|N_k|\!)\} \\
B_0' = B_0 \\
B_1' = \mathtt{0} \mid B_1
\end{array}
$$

We have $(\!|B_0'|\!) = P_0$ *directly and* $(\!|B_1'|\!) = P_1$ *by Lemma A.1. By* TMA *structure equivalence we have* $B_0 \equiv B_0'$ *and* $B_1 \equiv B_1'$. *Thus* $(\!|B_0'|\!) \overset{\mathcal{A}}{\hookrightarrow} (\!|B_1'|\!)$ *by* RREW.

(AAMB): *Here simply use the induction hypothesis and then instantiate the rule* $\mathbf{active}\{\, \mathring{\mathtt{P}}\ \mathbf{in}\ \mathring{\mathtt{a}}[\mathring{\mathtt{P}}] \,\}$ *in* RACT *by* $\rho = \{\mathring{\mathtt{a}} \mapsto n\}$ *where* n *is the ambient name obtained from the assumptions. Here we have to verify that* $n \neq \bullet$ *which is true for* $\bullet$ *is forbidden to be used by* TMA *processes.*

(ANU): *Again use the induction hypothesis and verify that the* ν*-bound name* n *is not in* $\mathsf{fn}(\mathcal{A}) = \{\mathsf{in}, \mathsf{out}, \mathsf{open}, \mathtt{[]}\}$. *This is satisfied for these name are excluded from* AName. *Then use* RNU *to prove the claim.*

(APAR): *Use the induction hypothesis and* RPAR *to prove the claim.*

(ASTR): *Use the induction hypothesis and* RSTR *to prove the claim.*

The following proposition is the right-to-left implication of Thm. 3.3.

PROPOSITION A.4. *Let* $(\!|B_0|\!) \overset{\mathcal{A}}{\hookrightarrow} P_1$. *Then there exists* B_1 *such that* $(\!|B_1|\!) \equiv P_1$ *and* $B_0 \to B_1$.

PROOF. *By induction on the derivation of* $(\!|B_0|\!) \overset{\mathcal{A}}{\hookrightarrow} P_1$. *Let it be derived by*

(RREW): *using the rule*

1. $\mathbf{rewrite}\{\, \mathring{P} \hookrightarrow \mathring{Q} \,\} = \mathbf{rewrite}\{\, \mathring{\mathtt{a}}[\mathsf{in}\ \mathring{\mathtt{b}}.\mathring{\mathtt{P}} \mid \mathring{\mathtt{Q}}] \mid \mathring{\mathtt{b}}[\mathring{\mathtt{R}}] \hookrightarrow \mathring{\mathtt{b}}[\mathring{\mathtt{a}}[\mathring{\mathtt{P}} \mid \mathring{\mathtt{Q}}] \mid \mathring{\mathtt{R}}] \,\}$. *We also know that there is some instantiation* ρ *with all the variables mentioned by the rule in its range. We define* $x = [\![\mathring{\mathtt{a}}]\!]_\rho$, $y = [\![\mathring{\mathtt{b}}]\!]_\rho$, $P_0' = [\![\mathring{\mathtt{P}}]\!]_\rho$, $P_1' = [\![\mathring{\mathtt{Q}}]\!]_\rho$, $P_2' = [\![\mathring{\mathtt{R}}]\!]_\rho$. *Now we can deduce that* $(\!|B_0|\!) = x[\mathsf{in}\ y.P_0' \mid P_1'] \mid y[P_2']$ *and* $P_1 = y[x[P_0' \mid P_1'] \mid P_2']$. *Now there have to be* B_0', B_1', B_2' *such that* $(\!|B_0'|\!) = P_0'$, $(\!|B_1'|\!) = P_1'$,

$(\!|B_2'|\!) = P_2'$, *and* $B_0 \equiv x[\mathsf{in}\ y.B_0' \mid B_1'] \mid y[B_2']$ *It holds that both* x *and* y *are in* AName *because (1)* ρ *can not map a name variable to* $\bullet$ *and (2)* $\mathsf{in}, \mathsf{out}, \mathsf{open}, []$ *can not appear in* B_0. *Now we just take* $B_1 = y[x[B_0' \mid B_1'] \mid B_2']$ *and thus we have* $(\!|B_1|\!) = P_1$. *Finally we proof* $B_0 \to B_1$ *by* AIn *and* AStr.

2. *Proof for the other three rules (*out, open, *and the communication one) is similar as case 1.*

(RAct): *using the rule* **active**$\{\mathring{P}\ \mathbf{in}\ \mathring{a}[\mathring{P}]\}$. *Denote* $x = [\![\mathring{a}]\!]_\rho$. *In this case we have that there are some* P *and* Q *such that* $P \overset{\mathcal{A}}{\hookrightarrow} Q$. *We also have that* $(\!|B_0|\!) = x[P]$ *and* $P_1 = x[Q]$. *Thus we see that there is some* B_0' *such that* $(\!|B_0'|\!) = P$ *and* $B_0 = x[B_0']$. *It also implies that* $x \in$ AName. *Thus we obtain* $(\!|B_0'|\!) \overset{\mathcal{A}}{\hookrightarrow} Q$ *and by the induction hypothesis we have that there exists* B_1' *such that* $(\!|B_1'|\!) \equiv Q$ *and* $B_0' \to B_1'$. *Take* $B_1 = x[B_1']$. *We have* $(\!|B_1|\!) = x[(\!|B_1'|\!)] \equiv x[Q] = P_1$. *Finally* $B_0 \to B_1$ *by* AAmb.

(RNu): *Thus there are* x, P, *and* Q, *such that* $(\!|B_0|\!) = \nu(x).P$ *and* $P_1 = \nu(x).Q$, *and* $P \overset{\mathcal{A}}{\hookrightarrow} Q$. *Here we see that* $x \in$ AName *and thus* $x \notin \mathsf{fn}(\mathcal{A})$. *From* $(\!|B_0|\!) = \nu(x).P$ *we can conclude that there are some* W *and* B_0' *such that* $B_0 = (\nu n : W)B_0'$ *and* $(\!|B_0'|\!) = P$. *Thus we have* $(\!|B_0'|\!) \overset{\mathcal{A}}{\hookrightarrow} Q$ *and by the induction hypothesis we obtain that there exists* B_1' *such that* $(\!|B_1'|\!) \equiv Q$ *and* $B_0' \to B_1'$. *Let us take* $B_1 = (\nu x : W)B_1'$. *Now* $(\!|B_1|\!) = \nu(x).(\!|B_1'|\!) \equiv \nu(x).Q = P_1$. *Finally* $B_0 \to B_1$ *by* ANu.

(RPar): *Proof is similar to case* RNu.

(RStr): *The problem to deal with in this case is the difference in structural equivalences of* Meta✶ *and* TMA, *in particular, the* Meta✶ *rule present which allows a* ν*-binder to skip an arbitrary action. Against that,* TMA *allows* ν*-binders to skip ambient boundaries only. For example for* $B_0 = ().(\nu \mathsf{a} : W)\mathsf{in}\ \mathsf{a}.0$ *and* $B_1 = (\nu \mathsf{a} : W)().\mathsf{in}\ \mathsf{a}.0$ *we have* $(\!|B_0|\!) \equiv (\!|B_1|\!)$ *in* Meta✶ *but not* $B_0 \equiv B_1$ *in* TMA. *The key observation here is that whenever in* Meta✶ *some rewriting is inferred by* RStr *using* Meta✶ *structural equivalence in a way that is not allowed in* TMA, *then the same rewriting statement can be inferred in* Meta✶ *using a derivation that uses structural equivalence only in a* TMA*-compatible way. Then rest of the proof is a simple application of the induction hypothesis.*

A.2 The Proof of Thm. 3.8

We use the following names for the typing rules of TMA which are assigned only to those rules from Fig. 9 that consider processes in the left-right and top-down order: TRep, TCap, TNul, TAmb, TPar, TNu, TOut, TIn. Similarly we name the Meta✶ typing rules which consider processes from Fig.5 as follows: SNul, SPar, SNu, SAct, SRep.

The following proposition is the left-to-right implication of Thm. 3.8.

Proposition A.6. $\mathsf{fn}(B) \subseteq \mathsf{dom}(E)\ \&\ E \vdash B : T \Rightarrow \vdash (\!|B|\!) : (\!|\mathsf{typeinfo}(E, B, T)|\!)$

PROOF. *Let* $\pi = \{\!|(\mathsf{benv}_E \cup \mathsf{nenv}_B, \mathsf{cenv}_B, T)|\!\}$. *Proof* $\vdash (\!|B|\!) : \pi$ *by induction on the derivation of* $E \vdash B : T$. *Let it be derived by*

(TREP): *Then there is* B_0 *such that* $B = !B_0$ *and* $E \vdash B_0 : T$. *By the induction hypothesis we obtain* $\pi_0 = \{\!|(\mathsf{benv}_E \cup \mathsf{nenv}_{B_0}, \mathsf{cenv}_{B_0}, T)|\!\}$ *such that* $\vdash (\!|B_0|\!) : \pi_0$. *Now obviously* $\mathsf{nenv}_{B_0} = \mathsf{nenv}_B$ *and* $\mathsf{cenv}_{B_0} = \mathsf{cenv}_B$. *Thus we have that* π *and* π_0 *are equal. Thus we can derive* $\vdash !(\!|B_0|\!) : \pi$ *by* SREP. *Hence the claim* $\vdash (\!|!B_0|\!) : \pi$ *holds by the definition of process encoding.*

(TNUL): *Obvious.*

(TAMB): *Here we have* $B = n[B_0]$ *and by the induction hypothesis we obtain* $\pi_0 = \{\!|(\mathsf{benv}_E \cup \mathsf{nenv}_{B_0}, \mathsf{cenv}_{B_0}, T')|\!\}$. *Let* $n = a^i$. *We have* $\mathsf{nenv}_B = \mathsf{nenv}_{B_0}$ *and* $\mathsf{cenv}_B = \mathsf{cenv}_{B_0}$ *thus* π *and* π_0 *differ only in the root nodes. Let* $\chi = \mathsf{nodeof}_I(T)$ *and* $\chi_0 = \mathsf{nodeof}_I(T')$. *It is clear that* χ *is the root of* $\pi = \langle G, \chi\rangle$ *and* χ_0 *of* $\pi_0 = \langle G, \chi_0\rangle$. *Moreover it is easy to see that* $a \in \mathsf{namesof}_I(\mathsf{Amb}[T'])$ *and thus* $(\chi \xrightarrow{a[]} \chi_0) \in G$. *From the induction hypothesis we have obtained that* $\vdash (\!|B_0|\!) : \langle G, \chi_0\rangle$ *and thus by* SACT *we have* $\vdash n[(\!|B_0|\!)] : \langle G, \chi\rangle$. *Thus the claim holds.*

(TCAP): *Similar as the case* TAMB.

(TPAR): *Here we have* $B = B_0 \mid B_1$ *and by the induction hypothesis we obtain two corresponding shape predicates* $\pi_0 = \{\!|(\mathsf{benv}_E \cup \mathsf{nenv}_{B_0}, \mathsf{cenv}_{B_0}, T)|\!\}$ *and* $\pi_1 = \{\!|(\mathsf{benv}_E \cup \mathsf{nenv}_{B_1}, \mathsf{cenv}_{B_1}, T)|\!\}$. *It is clear that* $\mathsf{nenv}_{B_0} \subseteq \mathsf{nenv}_B$ *and* $\mathsf{cenv}_{B_0} \subseteq \mathsf{cenv}_B$ *and thus the shape graph of* π_0 *is a subgraph of the shape graph of* π. *Thus because* $\vdash (\!|B_0|\!) : \pi_0$ *we have also* $\vdash (\!|B_0|\!) : \pi$. *Similarly also* $\vdash (\!|B_1|\!) : \pi$. *Hence the claim holds.*

(TNU): *Here* $B = (\nu n : W)B_0$ *and by the induction hypothesis gives us corresponding* $\pi_0 = \{\!|(\mathsf{benv}_{E_0} \cup \mathsf{nenv}_{B_0}, \mathsf{cenv}_{B_0}, T)|\!\}$ *where* $E_0 = E[n \mapsto W]$. *Also it has to be the case that* $W = \mathsf{Amb}[T']$ *for some* T'. *Thus we can see that* $\mathsf{benv}_{E_0} \cup \mathsf{nenv}_{B_0} = \mathsf{benv}_E \cup \mathsf{nenv}_B$ *and because also* $\mathsf{cenv}_{B_0} = \mathsf{cenv}_B$ *we obtain that* $\pi_0 = \pi$. *Hence* $\vdash (\!|B|\!) : \pi$ *by* SNU *because we have obtained* $\vdash (\!|B_0|\!) : \pi_0$ *by the induction hypothesis.*

(TOUT): *Here we have* $B = \texttt{<}N_1, \ldots, N_k\texttt{>}$ *and* $T = W_1 \otimes \cdots \otimes W_k$ *with* $E \vdash N_i : W_i$. *It is not hard to prove for* N_i *by induction on its structure that there exists* $\mu_i \in \mathsf{msgs}_I(W_i)$ *such that* $\vdash (\!|N_i|\!) : \mu_i$. *Let* $\pi = \langle G, \chi\rangle$. *We see also that* $\chi = \mathsf{nodeof}_I(T)$. *We can see that* $(\chi \xrightarrow{\texttt{<}\mu_1,\ldots,\mu_k\texttt{>}} \chi) \in G$ *and thus* $\vdash \texttt{<}(\!|N_1|\!), \ldots, (\!|N_k|\!)\texttt{>}.0 : \pi$. *Hence the claim holds.*

(TIN): *We follow case* TNU *and we have* $B = (n_1 : W_1, \ldots, n_k : W_k).B_0$ *and* $E_0 = E[n_1 \mapsto W_1, \ldots, n_k \mapsto W_k]$. *Let* $a_1^i = n_1$ *and so on. In this case we have* $\mathsf{nenv}_{B_0} = \mathsf{nenv}_B$ *and* $\mathsf{benv}_{E_0} = \mathsf{benv}_E \cup \{a_1 \mapsto W_1, \ldots, a_k \mapsto W_k\}$ *and* $\mathsf{cenv}_B = \mathsf{cenv}_{B_0} \cup \{a_1 \mapsto W_1, \ldots, a_k \mapsto W_k\}$. *Let* $\pi = \langle G, \chi\rangle$ *and* $\pi_0 = \langle G_0, \chi\rangle$ *where* π_0 *is from the induction hypothesis as in case* TNU. *But now we see that* G *contains more edges than* G_0 *for it additionally contains loops labeled with input-elements constructed from* a_i*'s. One of them is* $(\chi \xrightarrow{(a_1,\ldots,a_k)} \chi) \in G$. *By the induction hypothesis we have already obtained* $\vdash (\!|B_0|\!) : \langle G_0, \chi\rangle$ *and thus we have by* SACT *that* $\vdash (n_1, \ldots, n_k).(\!|B_0|\!) : \langle G, \chi\rangle$ *because of the presence of the above edge in* G. *Hence the claim holds.*

The following proposition is the right-to-left implication of Thm. 3.8.

PROPOSITION A.8. $\mathsf{fn}(B) \subseteq \mathsf{dom}(E)$ &$\vdash (\!|B|\!) : \{\!|\mathsf{typeinfo}(E, B, T)|\!\} \Rightarrow E \vdash B : T$

PROOF. *Let* $\pi = \{\!|(\mathsf{benv}_E \cup \mathsf{nenv}_B, \mathsf{cenv}_B, T)|\!\}$. *Proof* $E \vdash B : T$ *by induction on the derivation of* $\vdash (\!|B|\!) : \pi$. *Let it be derived by*

(SNUL): *Obvious.*

(SPAR): *Here we have* $(\!|B|\!) = P \mid Q$ *and* $\vdash P : \pi$ *and* $\vdash Q : \pi$. *There are* B_0 *and* B_1 *such that* $(\!|B_0|\!) = P$ *and* $(\!|B_1|\!) = Q$ *and* $B = B_0 \mid B_1$. *Now we have that* $\mathsf{nenv}_{B_0} \subseteq \mathsf{nenv}_B$ *and* $\mathsf{cenv}_{B_0} \subseteq \mathsf{cenv}_B$ *and similarly for* B_1. *The key observation here is that matching of* P *against* π *does not use edges of* π *which contain basic names that are not in* B_0. *Thus we can conclude that also* $\vdash (\!|B_0|\!) : \{\!|(\mathsf{benv}_E \cup \mathsf{nenv}_{B_0}, \mathsf{cenv}_{B_0}, T)|\!\}$ *and similarly for* B_1. *Now we can use the induction hypothesis and obtain* $E \vdash B_0 : T$ *and* $E \vdash B_1 : T$. *Hence the claim holds by* APAR.

(SNU): *Here* $(\!|B|\!) = \nu(n).P$ *and* $\vdash P : \pi$. *We can see that there are some* W *and* B_0 *such that* $(\!|B_0|\!) = P$ *and* $B = (\nu n : W)B_0$. *Let* $n = a^i$. *We also know that* $W = \mathsf{Amb}[T']$ *and thus* $\mathsf{nenv}_B(a) = W$. *Take* $E_0 = E[a^i \mapsto W]$. *Now we have* $\mathsf{cenv}_B = \mathsf{cenv}_{B_0}$ *and* $\mathsf{benv}_E \cup \mathsf{nenv}_B = \mathsf{benv}_{E_0} \cup \mathsf{nenv}_{B_0}$. *Thus we obtain* $\vdash (\!|B_0|\!) : \{\!|(\mathsf{benv}_{E_0} \cup \mathsf{nenv}_{B_0}, \mathsf{cenv}_{B_0}, T)|\!\}$. *Now we can use the induction hypothesis and we obtain that* $E_0 \vdash B_0 : T$. *Hence the claim holds by* ANU.

(SACT): *Here we have* $(\!|B|\!) = A.P$. *Let* $\pi = \langle G, \chi \rangle$. *Moreover there are some* α *and* χ_0 *such that* $(\chi \xrightarrow{\alpha} \chi_0) \in G$ *and* $\vdash A : \alpha$ *and* $\vdash P : \langle G, \chi_0 \rangle$. *Distinguish the following cases by the shape of* α*:*

- $\alpha = a[]$**:** *Thus for some* i *we have* $A = a^i[]$ *and also there is some* B_0 *such that* $(\!|B_0|\!) = P$ *and* $B = a^i[B_0]$. *Let* $T_0 = \mathsf{typeof}_I(\chi_0)$. *Then* $a \in \mathsf{namesof}_I(\mathsf{Amb}[T_0])$. *Now it has to be* $\mathsf{benv}_E(a) = \mathsf{Amb}[T_0]$ *because* a *is free in* B *(and* $A.P$*). Now by* S4 *and* $\mathsf{fn}(B) \subseteq \mathsf{dom}(E)$ *we have that* $E(a^i) = \mathsf{Amb}[T_0]$. *Thus* $E \vdash a^i : \mathsf{Amb}[T_0]$. *Now we know that* $\vdash P : \langle G, \chi_0 \rangle$ *and because* $\mathsf{nenv}_{B_0} = \mathsf{nenv}_B$ *and* $\mathsf{cenv}_{B_0} = \mathsf{cenv}_B$ *we have that* $\vdash (\!|B_0|\!) : \{\!|(\mathsf{benv}_E \cup \mathsf{nenv}_{B_0}, \mathsf{cenv}_{B_0}, T_0)|\!\}$. *Thus by the induction hypothesis we have that* $E \vdash B_0 : T_0$. *Thus the claim holds by* TAMB *because* $E \vdash a^i : \mathsf{Amb}[T_0]$.
- $\alpha = \mathsf{in}\ a$**:** *Then* $\chi_0 = \chi$ *and* $\mathsf{in}\ a \in \mathsf{moves}_I$. *Now* $A = \mathsf{in}\ a^i$ *for some* i *and there is some* B_0 *such that* $B = N.B_0$ *with* $N_* 0 = \mathsf{in}\ a^i.\cdots$. *Now there exists* $B' \equiv B$ *such that* $B' = \mathsf{in}\ a^i.B'_0$ *for some* B'_0. *Moreover we can choose* B'_0 *such that* $\mathsf{nenv}_B = \mathsf{nenv}_{B'_0}$ *and* $\mathsf{cenv}_B = \mathsf{cenv}_{B'_0}$. *Now we can see that* $\vdash (\!|B'_0|\!) : \{\!|(\mathsf{benv}_E \cup \mathsf{nenv}_{B'_0}, \mathsf{cenv}_{B'_0}, T)|\!\}$ *and use the induction hypothesis to obtain that* $E \vdash B'_0 : T$ *and thus* $E \vdash B_0 : T$. *Now because* $\mathsf{in}\ a \in \mathsf{moves}_I$ *and* $\mathsf{fn}(B_0) \subseteq \mathsf{dom}(E)$ *we know that there is some* T' *such that* $E(a^i) = \mathsf{Amb}[T']$. *Thus* $E \vdash \mathsf{in}\ a^i : \mathsf{Cap}[T]$. *Hence the claim holds by* ACAP.
- $\alpha = \mathsf{out}\ a$**:** *As the case for "*$\mathsf{in}\ a$*".*

$\alpha = \mathsf{open}\ a$**:** *As the case for "*$\mathsf{in}\ a$*" except* $\mathsf{open}\ a \in \mathsf{opens}_I(T)$ *and thus directly* $E(a^i) = \mathsf{Amb}[T]$ *(and not* T'*).*

$\alpha = a$**:** *Once again like the case for "*$\mathsf{in}\ a$*" but this time* $a \in \mathsf{opens}_I(T)$ *and thus directly* $E(a^i) = \mathsf{Cap}[T]$*.*

$\alpha = \texttt{<}\mu_1, \ldots, \mu_k\texttt{>}$**:** *Here we have that* $\alpha \in \mathsf{allowedin}_I(T)$ *and thus* $\chi = \chi_0$*. Thus* $T = W_1 \otimes \cdots \otimes W_k$ *and* $\mu_i \in W_i$*. Because* $\vdash A : \alpha$ *we have that* $A = \texttt{<}M_1, \ldots, M_k\texttt{>}$ *for some* $M_1, \ldots, M_k$ *with* $\vdash M_i : \mu_i$*. Now we know that there are some* $N_1, \ldots, N_k$ *such that* $(\!|N_i|\!) = M_i$ *and* $B = \texttt{<}N_1, \ldots, N_k\texttt{>}$*. Here it also has to be the case that* $P = \mathsf{0}$ *because* $(\!|B|\!) = A.P$*. Now it is easy to prove that* $E \vdash N_i : W_i$*. The proof is direct when* $\mu_i = a$ *and it is by induction on the structure of* N_i *when* $\mu_i = \Phi\!*$*. Hence the claim holds by* AOut*.*

$\alpha = (a_1, \ldots, a_k)$**:** *Here we have that* $\chi_0 = \chi$ *and* $\alpha \in \mathsf{comms}_I(T)$*. Thus* $T = W_1 \otimes \cdots \otimes W_k$ *and* $\mathsf{cenv}_B(a_i) = W_i$*. Let* $n_1 = a_1^i$ *and so on. Now we know that there is some* B_0 *such that* $B = (n_1 : W_1, \ldots, n_k : W_k).B_0$ *and* $(\!|B_0|\!) = P$*. Take* $E_0 = E[n_1 \mapsto W_1, \ldots, n_k \mapsto W_k]$*. Here it holds that* $\mathsf{benv}_{E_0} = \mathsf{benv}_E \cup \{n_1 \mapsto W_1, \ldots, n_k \mapsto W_k\}$ *and that* $\mathsf{cenv}_B = \mathsf{cenv}_{B_0} \cup \{n_1 \mapsto W_1, \ldots, n_k \mapsto W_k\}$*. Let* $\pi_0 = (\!|(\mathsf{benv}_{E_0} \cup \mathsf{nenv}_{B_0}, \mathsf{cenv}_{B_0}, T)|\!)$*. Now* π_0 *differs from* π *only in that* π *has additional self loops labeled by input-elements constructed from* a_i*'s. We know that* $\vdash P : \pi$*. Now it can be seen that* P *when matching against* π *does not used the above edges not present in* π_0 *because none of* a_i*'s can be input-bound in* P*. Thus also* $\vdash P : \pi_0$ *that is* $\vdash (\!|B_0|\!) : \pi_0$*. Now by the induction hypothesis we obtain that* $E_0 \vdash B_0 : T$*. Hence the claim holds by* TIn*.*

otherwise: *It follows from the construction of* π *that all possibilities are included in the above cases.*

(SRep): *Simply apply the induction hypothesis.*

A.3 The Proof of Thm. 4.2

In the proof of Thm. 4.2 we refer to some notions defined in the Poly✶ technical report [MW04, App. B], namely, the notion of *pre-principality* of the shape predicate π for the process P.

Definition A.10. *For* χ*,* a*, and the shape predicate* $\pi = \langle G, \chi' \rangle$ *write*

- $\mathsf{isanodeunder}_\pi(\chi, \star)$ *when* $(\chi' \xrightarrow{\alpha_0} \cdots \xrightarrow{\alpha_k} \chi) \in G$ *and no* α_i *contains* `[]`
- $\mathsf{isanodeunder}_\pi(\chi, a)$ *when* $(\chi_0 \xrightarrow{[]\hat{}a} \chi_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_k} \chi) \in G$ *and no* α_i *contains* `[]`

The proof sketch of Thm. 4.2 follows.

Theorem A.11. *Let* B *be a* BA⁻ *process,* $(\mathcal{I}, \mathcal{R})$ *the result of* SABA *analysis of* B*, and let* π *be the* Poly✶ *principal typing of* $(\!|B|\!)$*. Then* $\mathcal{I}_\pi \subseteq \mathcal{I}$ *and* $\mathcal{R}_\pi \subseteq \mathcal{R}$*.*

Proof. *Suppose we have the following derivation of the principal typing of* $(\!|B|\!)$*, that is, that we have the sequence of shape predicates* $\pi_0, \ldots, \pi_k$ *such that*

- *every* π_i *is pre-principal for* $(\!|B|\!)$,
- π_0 *corresponds directly to the syntax tree of* $(\!|B|\!)$,
- π_{i+1} *is the same as* π_i *except one edge (either a flow or normal edge) and that* π_{i+1} *has at most one more node than* π_i, *and*
- $\pi_k = \pi$.

Then by induction on the derivation of the principal typing proof that for every $\pi_i = \langle G_i, \chi_i \rangle$ *all the following hold:*

1. $\mathsf{inamb}_{\pi_i}(a, \alpha)$ *implies* $(a, \alpha) \in \mathcal{I}$,
2. $(\chi_0 \xrightarrow{\boxed{\{a \mapsto b\}}} \chi_1) \in G_i$ *implies* $(a, b) \in \mathcal{R}$, *and*
3. *for all* a', b' *such that* $(\chi_0 \xrightarrow{\boxed{\{a \mapsto b\}}} \chi_1) \in G_i$ *and* $\mathsf{isanodeunder}_{\pi_i}(\chi_0, a')$ *and* $\mathsf{isanodeunder}_{\pi_i}(\chi_1, b')$ *it holds that* $(\forall \alpha : (a', \alpha) \in \mathcal{I} \Rightarrow (b', \alpha\tau) \in \mathcal{I})$ *where* $\tau = \{a \mapsto b\}$.

Properties 1 and 2 when applied to π *prove the claim.*