

Shape Types for Ambients with Communication Dependencies

Torben Amtoft and Joe Wells

Heriot-Watt University

Ambient Calculi

The Ambient Calculus is a process calculus designed by Cardelli & Gordon, and later extended and modified by many others, to model these notions:

- **Location:** All processes are located in *ambients* which can be nested, forming a tree.

Ambient Calculi

The Ambient Calculus is a process calculus designed by Cardelli & Gordon, and later extended and modified by many others, to model these notions:

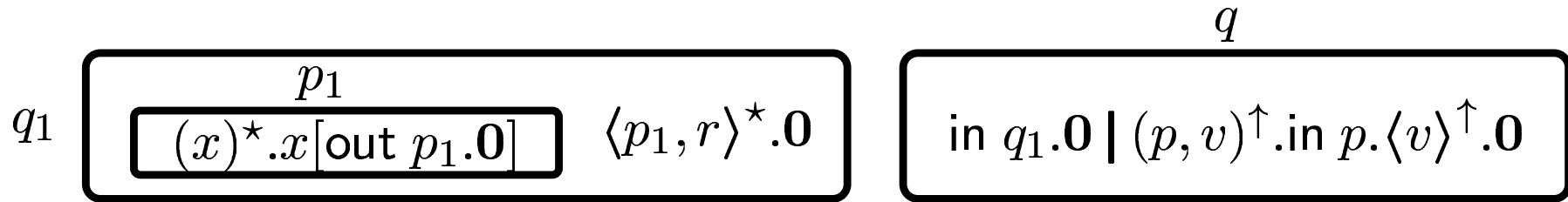
- **Location:** All processes are located in *ambients* which can be nested, forming a tree.
- **Mobility:** Ambients can move, making the tree dynamic.

Ambient Calculi

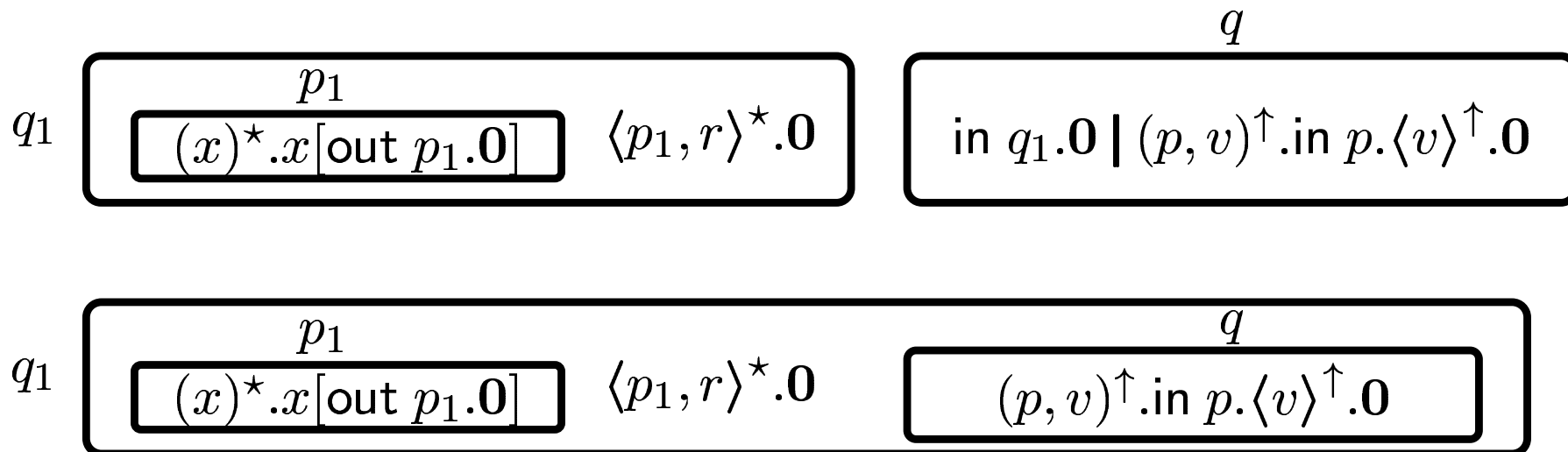
The Ambient Calculus is a process calculus designed by Cardelli & Gordon, and later extended and modified by many others, to model these notions:

- **Location:** All processes are located in *ambients* which can be nested, forming a tree.
- **Mobility:** Ambients can move, making the tree dynamic.
- **Communication:** Processes that are “*close*” to each other can exchange values.

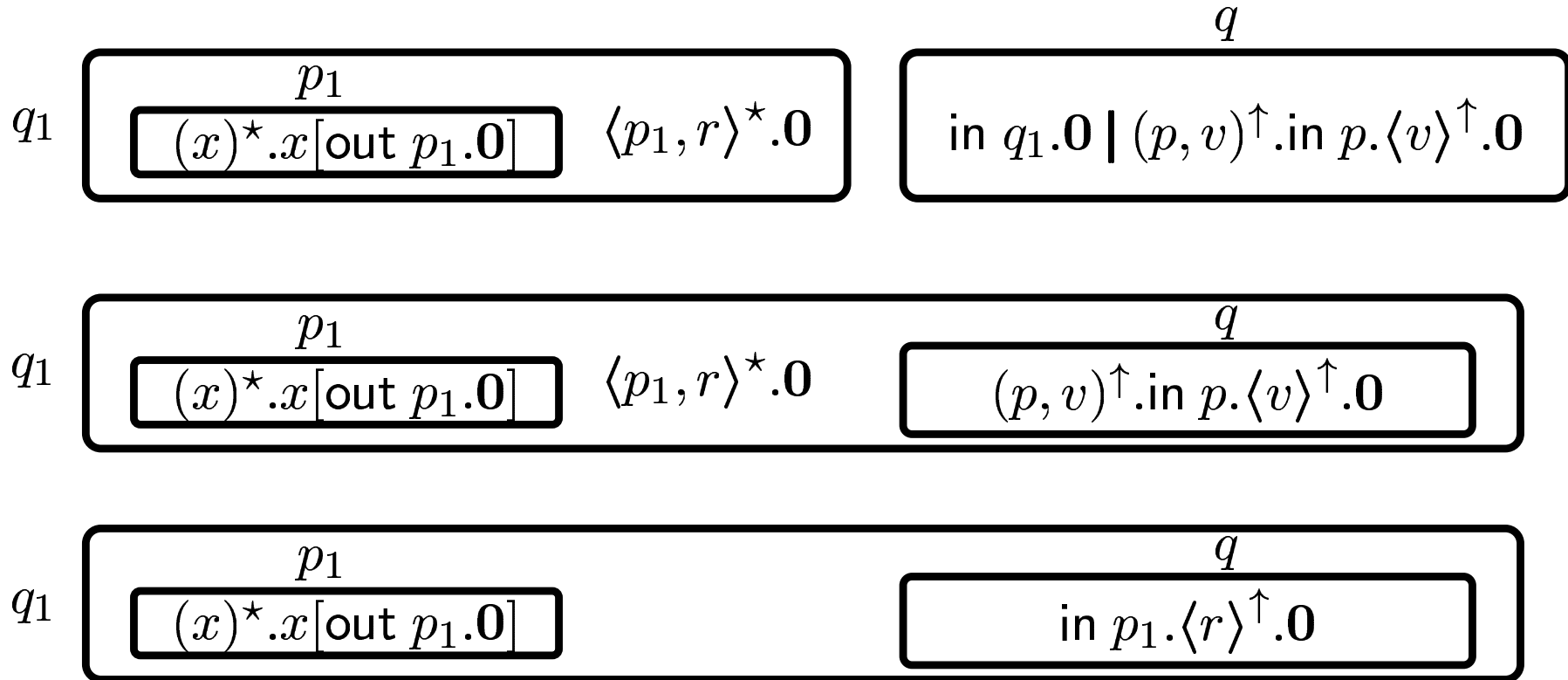
Example Rewrite Sequence



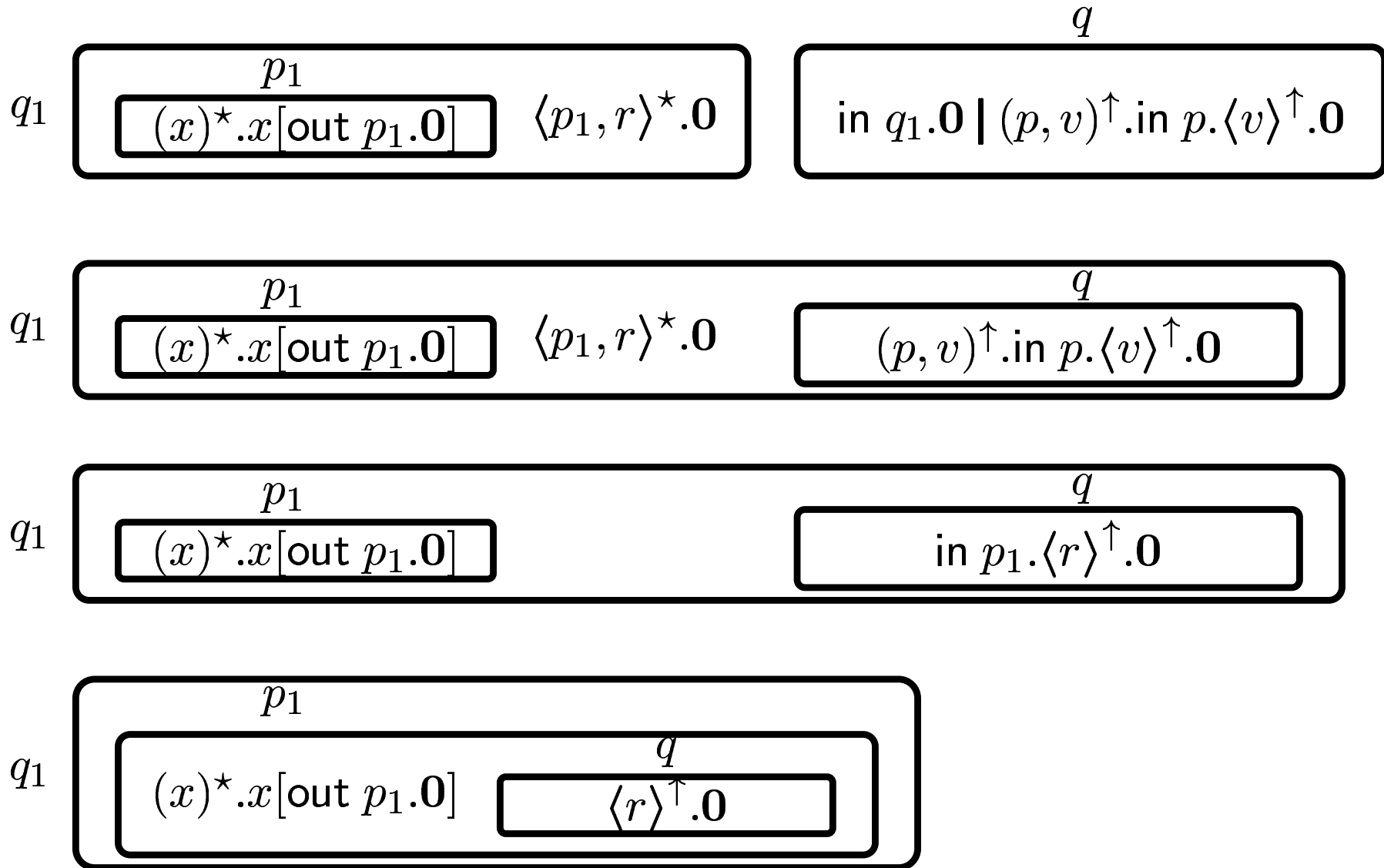
Example Rewrite Sequence



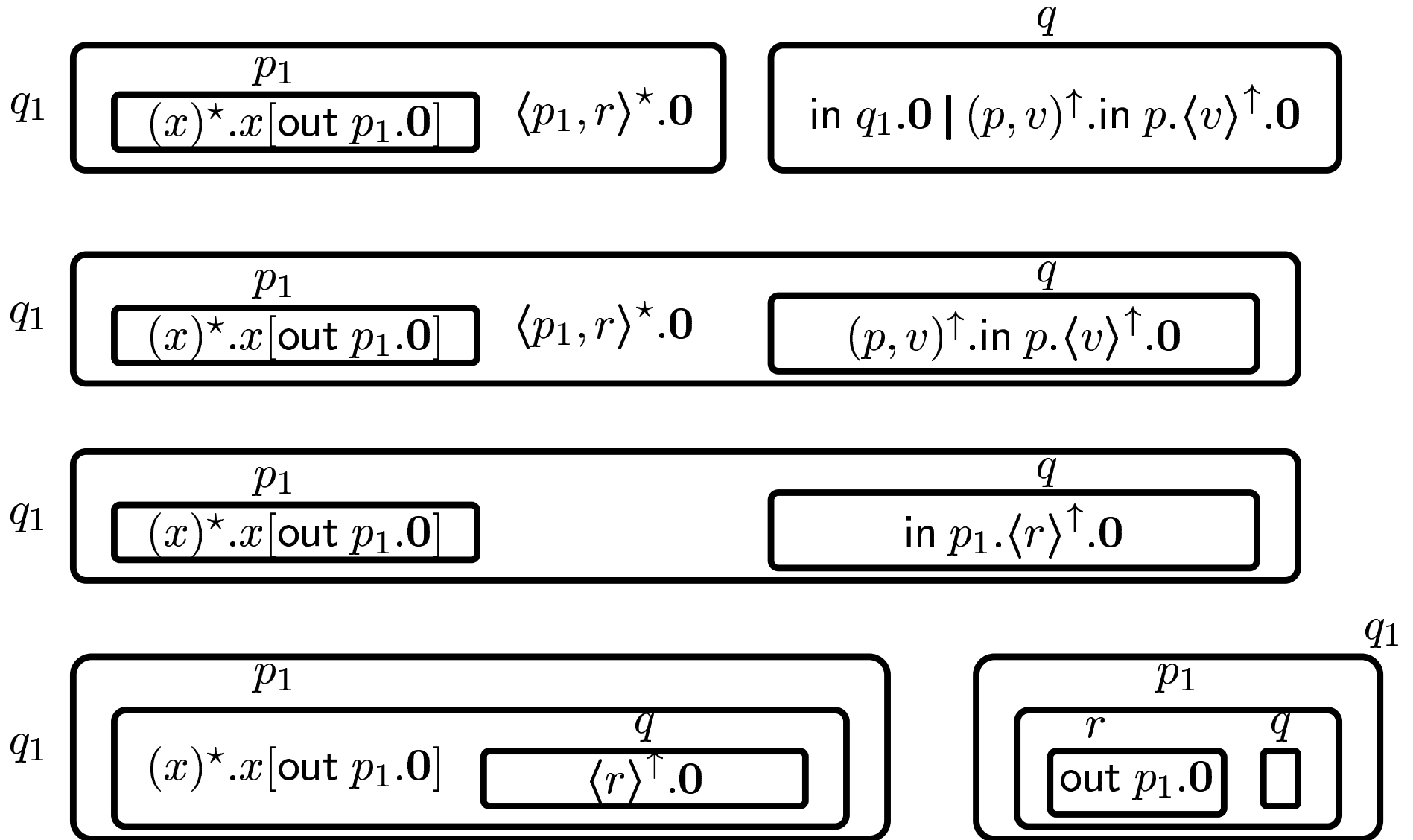
Example Rewrite Sequence



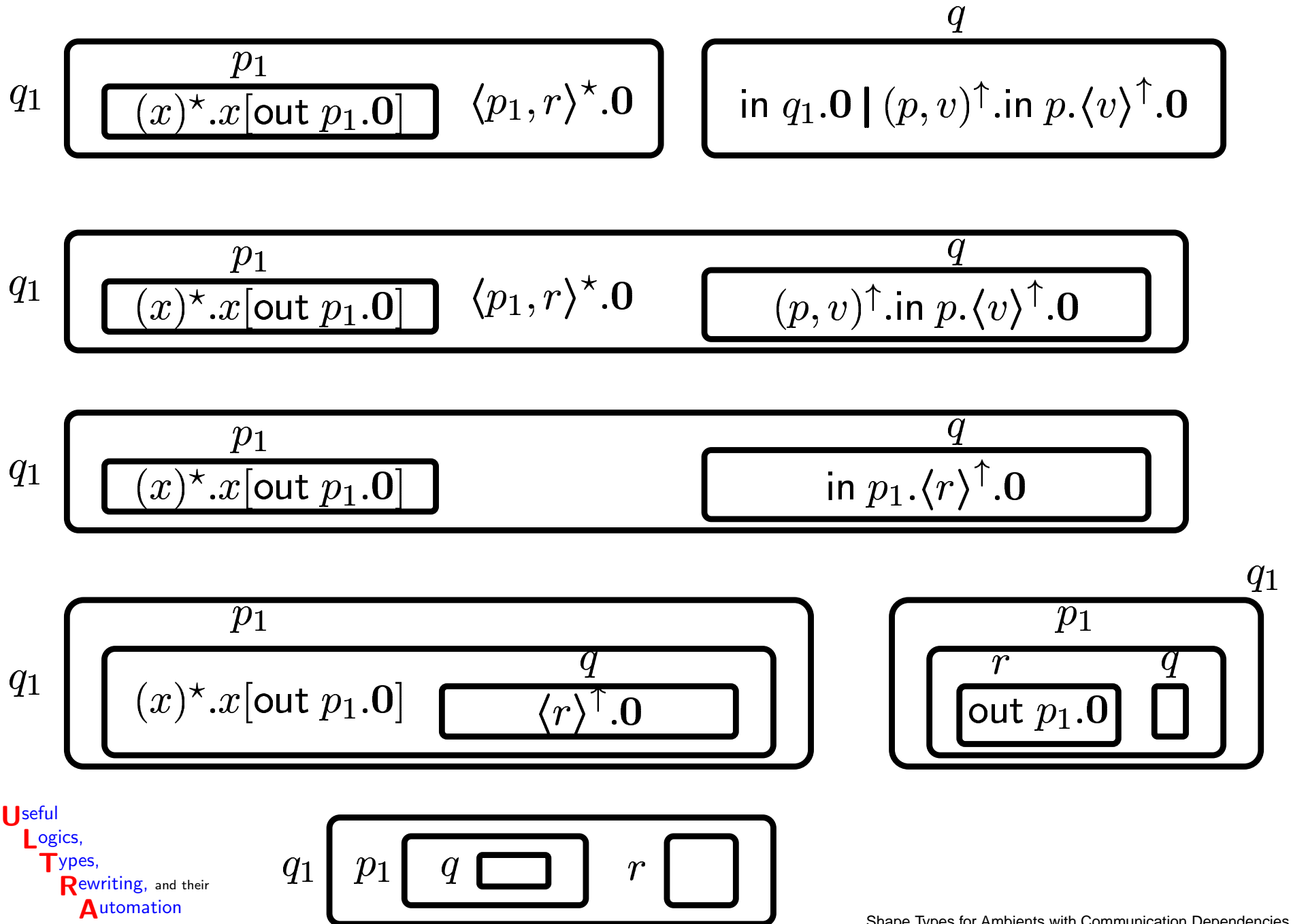
Example Rewrite Sequence



Example Rewrite Sequence



Example Rewrite Sequence



The Language Formalism

We use the extension of Boxed Ambients with open.

$$\lambda \in \text{Locs} ::= \star \mid \uparrow \mid \downarrow a$$
$$M \in \text{Exp} ::= a \mid c \mid \text{in } a \mid \text{out } a \mid \text{open } a \mid \epsilon \mid M_1.M_2$$
$$P, Q, R \in \text{Proc} ::= \mathbf{0} \mid P_1 \mid P_2 \mid !P \mid (\nu a).P$$
$$\mid M.P \mid a[P] \mid (\vec{a})^\lambda.P \mid \langle \vec{M} \rangle^\lambda.P$$

The Language Formalism

We use the extension of Boxed Ambients with open.

$$\lambda \in \text{Locs} ::= \star \mid \uparrow \mid \downarrow a$$

$$M \in \text{Exp} ::= a \mid c \mid \text{in } a \mid \text{out } a \mid \text{open } a \mid \epsilon \mid M_1.M_2$$

$$P, Q, R \in \text{Proc} ::= \mathbf{0} \mid P_1 \mid P_2 \mid !P \mid (\nu a).P \\ \mid M.P \mid a[P] \mid (\vec{a})^\lambda.P \mid \langle \vec{M} \rangle^\lambda.P$$

The semantics are defined by rewrite rules such as:

$$b[(\vec{a})^\uparrow.P \mid R] \mid \langle \vec{M} \rangle^\star.Q \longrightarrow b[P[\vec{a} := \vec{M}] \mid R] \mid Q$$

The Language Formalism

We use the extension of Boxed Ambients with open.

$$\lambda \in \text{Locs} ::= \star \mid \uparrow \mid \downarrow a$$

$$M \in \text{Exp} ::= a \mid c \mid \text{in } a \mid \text{out } a \mid \text{open } a \mid \epsilon \mid M_1.M_2$$

$$\begin{aligned} P, Q, R \in \text{Proc} ::= & \mathbf{0} \mid P_1 \mid P_2 \mid !P \mid (\nu a).P \\ & \mid M.P \mid a[P] \mid (\vec{a})^\lambda.P \mid \langle \vec{M} \rangle^\lambda.P \end{aligned}$$

The semantics are defined by rewrite rules such as:

$$b[(\vec{a})^\uparrow.P \mid R] \mid \langle \vec{M} \rangle^\star.Q \longrightarrow b[P[\vec{a} := \vec{M}] \mid R] \mid Q$$

Substitution may not always be well-defined:

$$(\text{in } a)[a := \text{out } b] \quad (b[a.\mathbf{0}])[a := \text{out } c] \quad (a[\text{in } b])[a := \epsilon]$$

The Language Formalism

We use the extension of Boxed Ambients with open.

$$\lambda \in \text{Locs} ::= \star \mid \uparrow \mid \downarrow a$$

$$M \in \text{Exp} ::= a \mid c \mid \text{in } a \mid \text{out } a \mid \text{open } a \mid \epsilon \mid M_1.M_2$$

$$P, Q, R \in \text{Proc} ::= \mathbf{0} \mid P_1 \mid P_2 \mid !P \mid (\nu a).P \\ \mid M.P \mid a[P] \mid (\vec{a})^\lambda.P \mid \langle \vec{M} \rangle^\lambda.P$$

The semantics are defined by rewrite rules such as:

$$b[(\vec{a})^\uparrow.P \mid R] \mid \langle \vec{M} \rangle^\star.Q \longrightarrow b[P[\vec{a} := \vec{M}] \mid R] \mid Q$$

Substitution may not always be well-defined:

$$(\text{in } a)[a := \text{out } b] \quad (b[a.\mathbf{0}])[a := \text{out } c] \quad (a[\text{in } b])[a := \epsilon]$$

A type system should ensure at least well-definedness.

Previous Type Systems

Remember our example:

$$\begin{aligned} & q_1[\langle p_1, r \rangle^* . \mathbf{0} \mid p_1[(x)^* . x[\text{out } p_1 . \mathbf{0}]]] \\ & \mid q[\text{in } q_1 . \mathbf{0} \mid (p, v)^\uparrow . \text{in } p . \langle v \rangle^\uparrow . \mathbf{0}] \end{aligned}$$

Previous Type Systems

Remember our example:

$$\begin{aligned} & q_1[\langle p_1, r \rangle^* . \mathbf{0} \mid p_1[(x)^* . x[\text{out } p_1 . \mathbf{0}]]] \\ & \mid q[\text{in } q_1 . \mathbf{0} \mid (p, v)^\uparrow . \text{in } p . \langle v \rangle^\uparrow . \mathbf{0}] \end{aligned}$$

This can be typed by the system for Boxed Ambients (in the spirit of Cardelli & Gordon’s original) because in each ambient the “topic of conversation” is well-defined (for each arity and direction):

$$\begin{aligned} \text{type of } v, x : T_1 &= \text{Amb}[\text{shh}, \text{shh}] \\ \text{type of } p, p_1 : T_2 &= \text{Amb}[T_1, \text{shh}] \\ \text{type of } q : T &= \text{Amb}[\text{shh}, T_1 \cup (T_2 \times T_1)] \end{aligned}$$

Previous Type Systems

Remember our example:

$$\begin{aligned} & q_1[\langle p_1, r \rangle^* . \mathbf{0} \mid p_1[(x)^* . x[\text{out } p_1 . \mathbf{0}]]] \\ & \mid q[\text{in } q_1 . \mathbf{0} \mid (p, v)^\uparrow . \text{in } p . \langle v \rangle^\uparrow . \mathbf{0}] \end{aligned}$$

This can be typed by the system for Boxed Ambients (in the spirit of Cardelli & Gordon’s original) because in each ambient the “topic of conversation” is well-defined (for each arity and direction):

$$\begin{aligned} \text{type of } v, x : T_1 &= \text{Amb}[\text{shh}, \text{shh}] \\ \text{type of } p, p_1 : T_2 &= \text{Amb}[T_1, \text{shh}] \\ \text{type of } q : T &= \text{Amb}[\text{shh}, T_1 \cup (T_2 \times T_1)] \end{aligned}$$

Note: It is unclear what such type assumptions really *mean*.

Example Needing Poly-morphism/variance

We extend the previous example process to have *two* possible execution paths:

$q_1[\langle p_1, r \rangle^*. \mathbf{0} \mid p_1[(x)^*. x[\text{out } p_1. \mathbf{0}]]]$ (*x must be a name*)

| $q_2[\langle p_2, \text{out } p_2 \rangle^*. \mathbf{0} \mid p_2[(x)^*. r[x. \mathbf{0}]]]$ (*x must be a capability*)

| $q[\text{in } q_1. \mathbf{0} \mid \text{in } q_2. \mathbf{0} \mid (p, v)^\uparrow. \text{in } p. \langle v \rangle^\uparrow. \mathbf{0}]$

Example Needing Poly-morphism/variance

We extend the previous example process to have *two* possible execution paths:

$$q_1[\langle p_1, r \rangle^* . \mathbf{0} \mid p_1[(x)^* . x[\text{out } p_1 . \mathbf{0}]]] \quad (x \text{ must be a name})$$
$$\mid q_2[\langle p_2, \text{out } p_2 \rangle^* . \mathbf{0} \mid p_2[(x)^* . r[x . \mathbf{0}]]] \quad (x \text{ must be a capability})$$
$$\mid q[\text{in } q_1 . \mathbf{0} \mid \text{in } q_2 . \mathbf{0} \mid (p, v)^\uparrow . \text{in } p . \langle v \rangle^\uparrow . \mathbf{0}]$$

So the type of v has to be *both* an ambient name *and* a capability. None of the existing type systems allow this.

Example Needing Poly-morphism/variance

We extend the previous example process to have *two* possible execution paths:

$$q_1[\langle p_1, r \rangle^* . \mathbf{0} \mid p_1[(x)^* . x[\text{out } p_1 . \mathbf{0}]]] \quad (x \text{ must be a name})$$
$$\mid q_2[\langle p_2, \text{out } p_2 \rangle^* . \mathbf{0} \mid p_2[(x)^* . r[x . \mathbf{0}]]] \quad (x \text{ must be a capability})$$
$$\mid q[\text{in } q_1 . \mathbf{0} \mid \text{in } q_2 . \mathbf{0} \mid (p, v)^\uparrow . \text{in } p . \langle v \rangle^\uparrow . \mathbf{0}]$$

So the type of v has to be *both* an ambient name *and* a capability. None of the existing type systems allow this.

Key observation: the topic of conversation within q depends on whether q is inside q_1 or inside q_2 .

Example Needing Poly-morphism/variance

We extend the previous example process to have *two* possible execution paths:

$$q_1[\langle p_1, r \rangle^* . \mathbf{0} \mid p_1[(x)^* . x[\text{out } p_1 . \mathbf{0}]]] \quad (x \text{ must be a name})$$
$$\mid q_2[\langle p_2, \text{out } p_2 \rangle^* . \mathbf{0} \mid p_2[(x)^* . r[x . \mathbf{0}]]] \quad (x \text{ must be a capability})$$
$$\mid q[\text{in } q_1 . \mathbf{0} \mid \text{in } q_2 . \mathbf{0} \mid (p, v)^\uparrow . \text{in } p . \langle v \rangle^\uparrow . \mathbf{0}]$$

So the type of v has to be *both* an ambient name *and* a capability. None of the existing type systems allow this.

Key observation: the topic of conversation within q depends on whether q is inside q_1 or inside q_2 .

To overcome some of the weaknesses of previous type systems for ambient calculi, we will now present a new type system.

The New Type System (1)

- The types represent upper bounds on the possible ambient nesting tree into which a process can evolve, e.g.:

$$\left(a[\text{in } b.\mathbf{0}] \mid b[\mathbf{0}] \right) \quad : \quad \left(a[\text{in } b] \mid b[a[\text{in } b]] \right)$$

The New Type System (1)

- The types represent upper bounds on the possible ambient nesting tree into which a process can evolve, e.g.:

$$\left(a[\text{in } b.\mathbf{0}] \mid b[\mathbf{0}] \right) \quad : \quad \left(a[\text{in } b] \mid b[a[\text{in } b]] \right)$$

- Types indicate the possible positions of capabilities, inputs, and outputs.

The New Type System (1)

- The types represent upper bounds on the possible ambient nesting tree into which a process can evolve, e.g.:

$$\left(a[\text{in } b.\mathbf{0}] \mid b[\mathbf{0}] \right) \quad : \quad \left(a[\text{in } b] \mid b[a[\text{in } b]] \right)$$

- Types indicate the possible positions of capabilities, inputs, and outputs.
- The types say nothing about the number of copies of a feature at a location.

The New Type System (2)

- There are singleton types of ambient names and explicit dependencies on communication, e.g.:

$$\left((x)^* . x[\mathbf{0}] \mid \langle a \rangle^* . \mathbf{0} \right) \quad : \quad \left(((x)^* \rightarrow x[\mathbf{0}]) \mid \langle a \rangle^* \mid a[\mathbf{0}] \right)$$

The New Type System (2)

- There are singleton types of ambient names and explicit dependencies on communication, e.g.:

$$\left((x)^* . x[\mathbf{0}] \mid \langle a \rangle^* . \mathbf{0} \right) \quad : \quad \left(((x)^* \rightarrow x[\mathbf{0}]) \mid \langle a \rangle^* \mid a[\mathbf{0}] \right)$$

- Sequential composition is replaced by parallel composition, except for inputs, e.g.:

$$\left(p[\text{in } q . \text{in } r . \mathbf{0}] \mid r[\mathbf{0}] \right) \quad : \quad \left(p[\text{in } q \mid \text{in } r] \mid r[p[\text{in } q \mid \text{in } r]] \right)$$

The New Type System (2)

- There are singleton types of ambient names and explicit dependencies on communication, e.g.:

$$\left((x)^* . x[\mathbf{0}] \mid \langle a \rangle^* . \mathbf{0} \right) \quad : \quad \left(((x)^* \rightarrow x[\mathbf{0}]) \mid \langle a \rangle^* \mid a[\mathbf{0}] \right)$$

- Sequential composition is replaced by parallel composition, except for inputs, e.g.:

$$\left(p[\text{in } q.\text{in } r.\mathbf{0}] \mid r[\mathbf{0}] \right) \quad : \quad \left(p[\text{in } q \mid \text{in } r] \mid r[p[\text{in } q \mid \text{in } r]] \right)$$

- The types merge distinct ambients at a location with the same name:

$$a[T_1] \mid a[T_2] \doteq a[T_1 \mid T_2]$$

The New Type System (3)

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

$$\left(!a[!in a.0] \right) \quad : \quad \left(letrec X = a[in a | X] in X \right)$$

The New Type System (3)

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

$$\left(!a[!in a.0] \right) \quad : \quad \left(letrec X = a[in a | X] in X \right)$$

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

Due to binders, their precise characterization is non-trivial (Glew, ESOP '02).

The New Type System (3)

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

$$\left(!a[!in a.0] \right) \quad : \quad \left(letrec X = a[in a | X] in X \right)$$

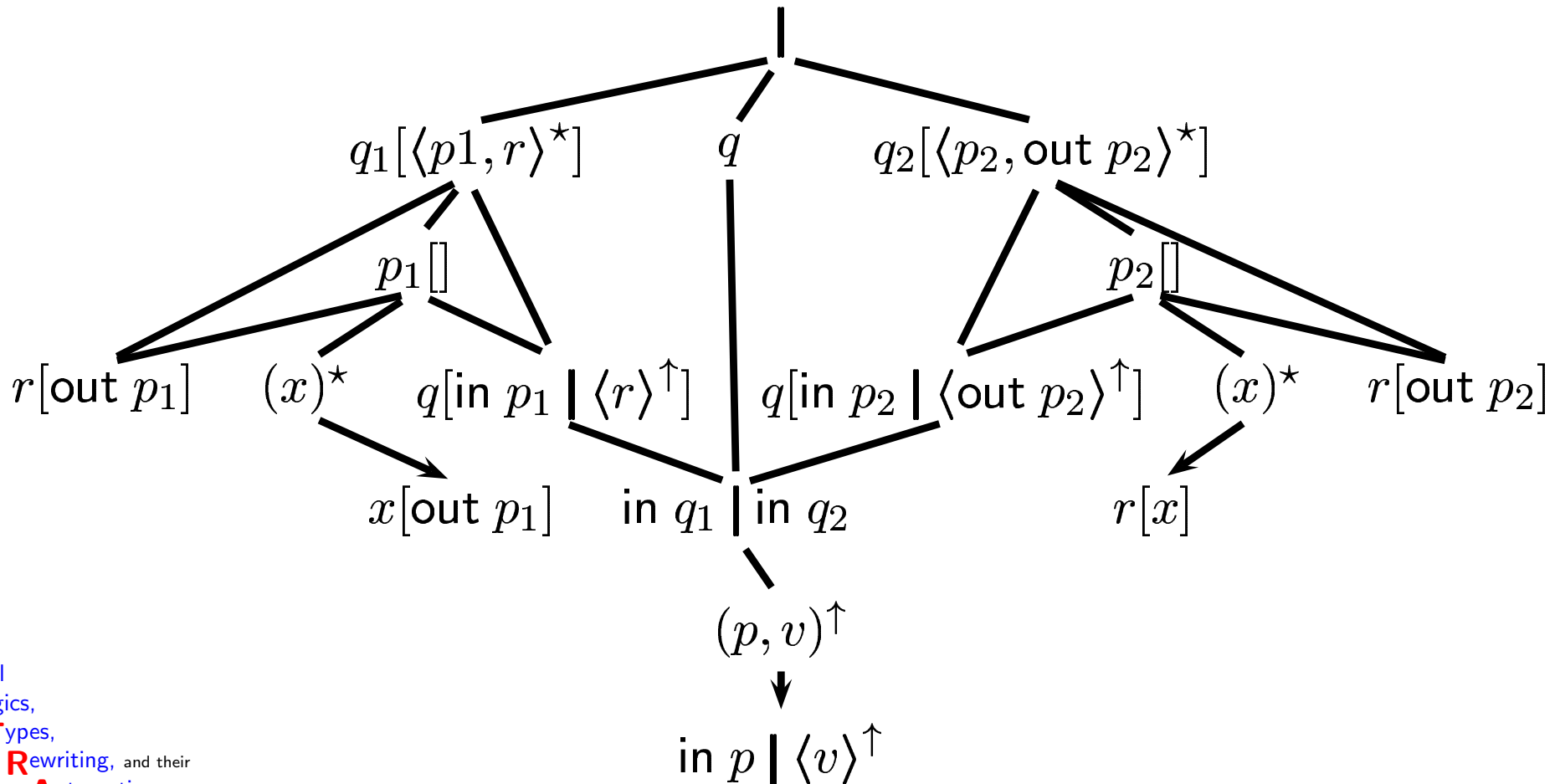
- We only consider types that can be given a finite term representation.

Due to binders, their precise characterization is non-trivial (Glew, ESOP '02).

- For our convenience, there is only one sort of types which is used for both messages (a.k.a. expressions) and processes.

Typing for Example Program

- $q_1[\langle p_1, r \rangle^* . \mathbf{0} \mid p_1[(x)^* . x[\text{out } p_1 . \mathbf{0}]]]$ (x must be a name)
- | $q_2[\langle p_2, \text{out } p_2 \rangle^* . \mathbf{0} \mid p_2[(x)^* . r[x . \mathbf{0}]]]$ (x must be a capability)
- | $q[\text{in } q_1 . \mathbf{0} \mid \text{in } q_2 . \mathbf{0} \mid (p, v)^\uparrow . \text{in } p . \langle v \rangle^\uparrow . \mathbf{0}]$



Subtyping and Closedness

There is an ordering \leq on types, with $T_1 \leq T_2$ meaning that T_1 is a more precise shape than T_2 .

Subtyping and Closedness

There is an ordering \leq on types, with $T_1 \leq T_2$ meaning that T_1 is a more precise shape than T_2 .

Parallel composition (“|”) is least upper bound w.r.t. that ordering.

Subtyping and Closedness

There is an ordering \leq on types, with $T_1 \leq T_2$ meaning that T_1 is a more precise shape than T_2 .

Parallel composition (“|”) is least upper bound w.r.t. that ordering.

We demand that types are closed under certain rules that simulate rewriting, such as:

$$a[T_1] \mid b[T_2] \leq T \text{ and } (\text{in } b) \leq T_1 \Rightarrow b[a[T_1] \mid T_2] \leq T$$

$$(\vec{a})^* \rightarrow T' \leq T \text{ and } \langle \vec{T} \rangle^* \leq T \Rightarrow T'[\vec{a} := \vec{T}] \leq T$$

(implying LHS well-defined)

To compute the closure, approximations are needed.

Type System

$$\frac{P_1 : T_1 \quad P_2 : T_2}{P_1 \mid P_2 : T_1 \mid T_2}$$

(very intuitive, as most rules)

Type System

$$\frac{P_1 : T_1 \quad P_2 : T_2}{P_1 \mid P_2 : T_1 \mid T_2}$$

(very intuitive, as most rules)

$$\frac{P : T}{!P : T}$$

(as no multiplicities)

Type System

$$\frac{P_1 : T_1 \quad P_2 : T_2}{P_1 \mid P_2 : T_1 \mid T_2}$$

(very intuitive, as most rules)

$$\frac{P : T}{!P : T}$$

(as no multiplicities)

$$\frac{M_1 : T_1 \quad M_2 : T_2}{M_1.M_2 : T_1 \mid T_2 \mid \text{action}}$$

(need to record it's not a name)

Type System

$$\frac{P_1 : T_1 \quad P_2 : T_2}{P_1 \mid P_2 : T_1 \mid T_2}$$

(very intuitive, as most rules)

$$\frac{P : T}{!P : T}$$

(as no multiplicities)

$$\frac{M_1 : T_1 \quad M_2 : T_2}{M_1.M_2 : T_1 \mid T_2 \mid \text{action}}$$

(need to record it's not a name)

$$\frac{M : T_0 \quad P : T}{M.P : T_0 \mid T}$$

Type System

$$\frac{P_1 : T_1 \quad P_2 : T_2}{P_1 \mid P_2 : T_1 \mid T_2}$$

(very intuitive, as most rules)

$$\frac{P : T}{!P : T}$$

(as no multiplicities)

$$\frac{M_1 : T_1 \quad M_2 : T_2}{M_1.M_2 : T_1 \mid T_2 \mid \text{action}}$$

(need to record it's not a name)

$$\frac{M : T_0 \quad P : T}{M.P : T_0 \mid T}$$

- **Subject reduction:** If $P \longrightarrow Q$ and $P : T$ (thus T closed), then $Q : T$.

Type System

$$\frac{P_1 : T_1 \quad P_2 : T_2}{P_1 \mid P_2 : T_1 \mid T_2} \quad (\text{very intuitive, as most rules})$$

$$\frac{P : T}{!P : T} \quad (\text{as no multiplicities})$$

$$\frac{M_1 : T_1 \quad M_2 : T_2}{M_1.M_2 : T_1 \mid T_2 \mid \text{action}} \quad (\text{need to record it's not a name})$$

$$\frac{M : T_0 \quad P : T}{M.P : T_0 \mid T}$$

- **Subject reduction:** If $P \longrightarrow Q$ and $P : T$ (thus T closed), then $Q : T$.
- **Safety:** If $P : T$ (thus T closed), then execution of P will never give rise to an ill-defined substitution.

Anomaly with open

Consider this term:

$$a[\text{in } b.\mathbf{0}] \mid b[\text{open } a.\mathbf{0}]$$

Ignoring the closedness requirement, we could give it this type:

$$a[\text{in } b] \mid b[\text{open } a]$$

Anomaly with open

Consider this term:

$$a[\text{in } b.\mathbf{0}] \mid b[\text{open } a.\mathbf{0}]$$

Ignoring the closedness requirement, we could give it this type:

$$a[\text{in } b] \mid b[\text{open } a]$$

To close this type, observe that a can go into b :

$$a[\text{in } b] \mid b[a[\text{in } b] \mid \text{open } a]$$

Anomaly with open

Consider this term:

$$a[\text{in } b.\mathbf{0}] \mid b[\text{open } a.\mathbf{0}]$$

Ignoring the closedness requirement, we could give it this type:

$$a[\text{in } b] \mid b[\text{open } a]$$

To close this type, observe that a can go into b :

$$a[\text{in } b] \mid b[a[\text{in } b] \mid \text{open } a]$$

Then a can be opened:

$$a[\text{in } b] \mid b[a[\text{in } b] \mid \text{open } a \mid \text{in } b]$$

Anomaly with open

Consider this term:

$$a[\text{in } b.0] \mid b[\text{open } a.0]$$

Ignoring the closedness requirement, we could give it this type:

$$a[\text{in } b] \mid b[\text{open } a]$$

To close this type, observe that a can go into b :

$$a[\text{in } b] \mid b[a[\text{in } b] \mid \text{open } a]$$

Then a can be opened:

$$a[\text{in } b] \mid b[a[\text{in } b] \mid \text{open } a \mid \text{in } b]$$

Now, one copy of b can go into another:

$$\dots \mid b[\dots \mid \text{in } b \mid b[\dots \mid \text{in } b]]$$

Anomaly with open

Consider this term:

$$a[\text{in } b.\mathbf{0}] \mid b[\text{open } a.\mathbf{0}]$$

Ignoring the closedness requirement, we could give it this type:

$$a[\text{in } b] \mid b[\text{open } a]$$

To close this type, observe that a can go into b :

$$a[\text{in } b] \mid b[a[\text{in } b] \mid \text{open } a]$$

Then a can be opened:

$$a[\text{in } b] \mid b[a[\text{in } b] \mid \text{open } a \mid \text{in } b]$$

Now, one copy of b can go into another:

$$\dots \mid b[\dots \mid \text{in } b \mid b[\dots \mid \text{in } b]]$$

This repeats forever. To close the type requires a recursive type:

$$a[\text{in } b] \mid \text{letrec } X = b[a[\text{in } b] \mid \text{open } a \mid \text{in } b \mid X] \text{ in } X$$

What to do about the anomaly?

- Ignore it. This is probably okay, but the types would look ugly.

What to do about the anomaly?

- **Ignore it.** This is probably okay, but the types would look ugly.
- **Avoid using open.** As is the recent trend, for instance in Boxed Ambients.

Note: the ability to dissolve an ambient will be crucial for new applications in modeling intracellular biological processes.

What to do about the anomaly?

- **Ignore it.** This is probably okay, but the types would look ugly.
- **Avoid using open.** As is the recent trend, for instance in Boxed Ambients.

Note: the ability to dissolve an ambient will be crucial for new applications in modeling intracellular biological processes.

- **Multiplicities.** Counting is not enough because the types “confuse the past with the future”, e.g., the count z must be ω to make this type closed:

$$(\text{open } a)^1 \mid a[(\text{in } b)^1]^1 \mid (\text{in } b)^z$$

What to do about the anomaly?

- **Ignore it.** This is probably okay, but the types would look ugly.
- **Avoid using open.** As is the recent trend, for instance in Boxed Ambients.

Note: the ability to dissolve an ambient will be crucial for new applications in modeling intracellular biological processes.

- **Multiplicities.** Counting is not enough because the types “confuse the past with the future”, e.g., the count z must be ω to make this type closed:

$$(\text{open } a)^1 \mid a[(\text{in } b)^1]^1 \mid (\text{in } b)^z$$

- **Union types.** Together with multiplicities, union types could work theoretically, but they are not feasible because each point in the possible future state space would likely become a separate type.

Embedding Cardelli & Gordon (POPL '99)

At least for ν -free programs, we can translate a typing.

$$a[\text{open } b.(y)^*. \text{in } y.\mathbf{0} \mid b[\langle c \rangle^*.\mathbf{0}]] \mid c[\mathbf{0}]$$

Embedding Cardelli & Gordon (POPL '99)

At least for ν -free programs, we can translate a typing.

$$a[\text{open } b.(y)^*. \text{in } y.\mathbf{0} \mid b[\langle c \rangle^*.\mathbf{0}]] \mid c[\mathbf{0}]$$

They assign $T_1 = \text{Amb}[\text{shh}]$ to c and y

They assign $T_2 = \text{Amb}[T_1]$ to a and b .

Embedding Cardelli & Gordon (POPL '99)

At least for ν -free programs, we can translate a typing.

$$a[\text{open } b.(y)^*. \text{in } y.0 \mid b[\langle c \rangle^*.0]] \mid c[0]$$

They assign $T_1 = \text{Amb}[\text{shh}]$ to c and y

They assign $T_2 = \text{Amb}[T_1]$ to a and b .

A mechanical translation converts the above into this type:

$$\begin{aligned} \text{letrec } X_C &= \text{in } \{a, b, c, y\} \mid \text{out } \{a, b, c, y\} \mid \text{action} \\ X_A &= a[X_1] \mid b[X_1] \mid c[X_0] \\ X_0 &= X_A \mid X_C \mid \text{open } \{c, y\} \\ X_1 &= X_A \mid X_C \mid \text{open } \{a, b\} \mid \langle c \rangle^* \mid \langle y \rangle^* \mid (y)^* \rightarrow X_1 \\ \text{in } X_0 \end{aligned}$$

Embedding Cardelli & Gordon (POPL '99)

At least for ν -free programs, we can translate a typing.

$$a[\text{open } b.(y)^*. \text{in } y.0 \mid b[\langle c \rangle^*.0]] \mid c[0]$$

They assign $T_1 = \text{Amb}[\text{shh}]$ to c and y

They assign $T_2 = \text{Amb}[T_1]$ to a and b .

A mechanical translation converts the above into this type:

$$\begin{aligned} \text{letrec } X_C &= \text{in } \{a, b, c, y\} \mid \text{out } \{a, b, c, y\} \mid \text{action} \\ X_A &= a[X_1] \mid b[X_1] \mid c[X_0] \\ X_0 &= X_A \mid X_C \mid \text{open } \{c, y\} \\ X_1 &= X_A \mid X_C \mid \text{open } \{a, b\} \mid \langle c \rangle^* \mid \langle y \rangle^* \mid (y)^* \rightarrow X_1 \\ &\text{in } X_0 \end{aligned}$$

This can probably be extended to Mobility Types
(Cardelli & Ghelli & Gordon).

Other Kinds of Poly-morphic/variant Analysis

- **Shape grammars** (Nielson & Nielson, POPL '00)
Returns a set of grammars such that at any step, the current process can be described by one of these grammars.
Very precise, but potentially also very expensive.

Other Kinds of Poly-morphic/variant Analysis

- **Shape grammars** (Nielson & Nielson, POPL '00)
Returns a set of grammars such that at any step, the current process can be described by one of these grammars.
Very precise, but potentially also very expensive.
- **Kleene Analysis** (Nielson & Nielson & Sagiv, ESOP '00)
Using 3-valued logic, estimates the possible shapes.
Trade-offs w.r.t. precision vs. costs.

Other Kinds of Poly-morphic/variant Analysis

- **Shape grammars** (Nielson & Nielson, POPL '00)
Returns a set of grammars such that at any step, the current process can be described by one of these grammars.
Very precise, but potentially also very expensive.
- **Kleene Analysis** (Nielson & Nielson & Sagiv, ESOP '00)
Using 3-valued logic, estimates the possible shapes.
Trade-offs w.r.t. precision vs. costs.
- **Abstract Interpretation** (Levi & Maffeis, SAS '01)
Keeps track of the context “one level up”.
Quite precise, and yet “only” polynomial (n^7).

Other Kinds of Poly-morphic/variant Analysis

- **Shape grammars** (Nielson & Nielson, POPL '00)
Returns a set of grammars such that at any step, the current process can be described by one of these grammars.
Very precise, but potentially also very expensive.
- **Kleene Analysis** (Nielson & Nielson & Sagiv, ESOP '00)
Using 3-valued logic, estimates the possible shapes.
Trade-offs w.r.t. precision vs. costs.
- **Abstract Interpretation** (Levi & Maffeis, SAS '01)
Keeps track of the context “one level up”.
Quite precise, and yet “only” polynomial (n^7).

None of the above-listed work handles communication, so none can show our example is safe.

Conclusion

We presented a type system for a variant ambient calculus that:

- Is poly-morphic/variant in that an ambient is analyzed differently for different interactions it enters into.

Conclusion

We presented a type system for a variant ambient calculus that:

- Is poly-morphic/variant in that an ambient is analyzed differently for different interactions it enters into.
- Has dependent typing where the analysis tracks which values are communicated and reacts accordingly.

Conclusion

We presented a type system for a variant ambient calculus that:

- Is poly-morphic/variant in that an ambient is analyzed differently for different interactions it enters into.
- Has dependent typing where the analysis tracks which values are communicated and reacts accordingly.

Future work includes:

- Writing a terminating algorithm for computing closure.

Conclusion

We presented a type system for a variant ambient calculus that:

- Is poly-morphic/variant in that an ambient is analyzed differently for different interactions it enters into.
- Has dependent typing where the analysis tracks which values are communicated and reacts accordingly.

Future work includes:

- Writing a terminating algorithm for computing closure.
- Investigating the relationship to other systems. For instance, it seems possible to embed the types into the logic of Cardelli & Ghelli (ESOP '01).

Conclusion

We presented a type system for a variant ambient calculus that:

- Is poly-morphic/variant in that an ambient is analyzed differently for different interactions it enters into.
- Has dependent typing where the analysis tracks which values are communicated and reacts accordingly.

Future work includes:

- Writing a terminating algorithm for computing closure.
- Investigating the relationship to other systems. For instance, it seems possible to embed the types into the logic of Cardelli & Ghelli (ESOP '01).
- Evaluating practical usefulness and feasibility.