

Mailbox Types for Unordered Interactions¹

Presentación por Jonathan Bekenstein

Materia optativa sobre Tipos Comportamentales y Contratos

2024

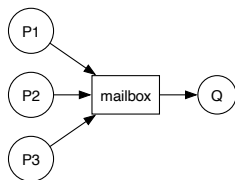


¹Ugo de'Liguoro, Luca Padovani (2018). <https://arxiv.org/abs/1801.04167>

Introducción

Buscamos modelar protocolos de comunicación en diferentes topologías de procesos concurrentes. La comunicación es **multi-party** y sucede mediante el uso de **mailboxes** no ordenados, en donde:

- ▶ procesos pueden escribir mensajes identificados por un *tag* con argumentos opcionales,
- ▶ procesos pueden consumir (leer) los mensajes en un orden arbitrario (*out-of-order* o *selective processing*).



Procesos bien tipados respetan el protocolo y no tienen deadlocks.

Mailbox calculus

Sintaxis: gramática

Process	$P, Q ::= \text{done}$	(termination)
	$\mathbf{X}[\bar{u}]$	(invocation)
	G	(guarded process)
	$u!\mathbf{m}[\bar{v}]$	(stored message)
	$P \mid Q$	(parallel composition)
	$(\nu a)P$	(mailbox restriction)

Guard	$G, H ::= \text{fail } u$	(runtime error)
	$\text{free } u.P$	(mailbox deletion)
	$u?\mathbf{m}(\bar{x}).P$	(selective receive)
	$G + H$	(guard composition)

\bar{u} denota la secuencia u_1, \dots, u_n

Mailbox calculus

Sintaxis: mensajes

Enviar mensajes

$u!\mathbf{m}[\bar{v}]$

Guarda un mensaje identificado con el tag \mathbf{m} y argumentos \bar{v} en el mailbox u .

Recibir mensajes

$u?\mathbf{m}(\bar{x}).P$

Consume selectivamente el mensaje con tag \mathbf{m} del mailbox u y continúa con P reemplazando \bar{x} por los argumentos del mensaje.

Mailbox calculus

Sintaxis: procesos

Invocación

$X[\bar{u}]$ representa la invocación de un proceso llamado X con parámetros \bar{u} . Asumimos que existe una definición global de procesos de la forma $X(\bar{x}) \triangleq P$.

Paralelo y restricción

$P \mid Q$ denota la composición paralela de procesos, y $(\nu a)P$ representa un mailbox a restringido al scope de P .

Terminación

Un proceso **done** representa un proceso terminado y que no realiza ninguna otra acción.

Mailbox calculus

Sintaxis: guardas

Guardas

Las guardas G y la composición de guardas $G + H$ nos permite modelar distintas “ramas” de ejecución en función del mensaje consumido del mailbox. Luego se usa exclusivamente la continuación de la guarda que consumió el mensaje.

Errores

fail u permite modelar un runtime error al recibir un mensaje inesperado.

Eliminar mailbox

free $u.P$ permite eliminar el mailbox u si ya no se va a utilizar y continuar la ejecución con P .

Mailbox calculus

Semántica operacional

Reglas de reducción

[R-READ]	$a!m[\bar{c}] \mid a?m(\bar{x}).P + G \rightarrow P\{\bar{c}/\bar{x}\}$
[R-FREE]	$(\nu a)(\mathbf{free} \ a.P + G) \rightarrow P$
[R-DEF]	$X[\bar{c}] \rightarrow P\{\bar{c}/\bar{x}\} \quad \text{if } X(\bar{x}) \triangleq P$
[R-PAR]	$P \mid R \rightarrow Q \mid R \quad \text{if } P \rightarrow Q$
[R-NEW]	$(\nu a)P \rightarrow (\nu a)Q \quad \text{if } P \rightarrow Q$
[R-STRUCT]	$P \rightarrow Q \quad \text{if } P \equiv P' \rightarrow Q' \equiv Q$

Relación de congruencia estructural

$$\begin{array}{lll} \mathbf{fail} \ a + G \equiv G & G + H \equiv H + G & G + (H + H') \equiv (G + H) + H' \\ \mathbf{done} \mid P \equiv P & P \mid Q \equiv Q \mid P & P \mid (Q \mid R) \equiv (P \mid Q) \mid R \\ & (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P & (\nu a)P \mid Q \equiv (\nu a)(P \mid Q) \quad \text{if } a \notin \text{fn}(Q) \end{array}$$

Ejemplo 1: Lock

$$\begin{aligned}\text{FreeLock}(self) &\triangleq \text{free } self.\text{done} \\ &\quad + self?\text{acquire}(owner).\text{BusyLock}[self, owner] \\ &\quad + self?\text{release}.\text{fail } self\end{aligned}$$
$$\text{BusyLock}(self, owner) \triangleq owner!\text{reply}[self] \mid self?\text{release}.\text{FreeLock}[self]$$
$$\text{User}(self, lock) \triangleq lock!\text{acquire}[self] \mid self?\text{reply}(l).(l!\text{release} \mid \text{free } self.\text{done})$$
$$(\nu lock)(\nu alice)(\nu carol)(\text{FreeLock}[lock] \mid \text{User}[alice, lock] \mid \text{User}[carol, lock])$$

Observaciones

- ▶ FreeLock consume de manera no determinística los mensajes **acquire**.
- ▶ User utiliza la referencia l para enviar el mensaje **release** ya que es ésta la referencia al mailbox que tiene la capacidad de procesar este mensaje.

Mailbox calculus

Caracterizaciones operacionales

Contextos de procesos

$$\mathcal{C} ::= [] \mid \mathcal{C} \mid P \mid P \mid \mathcal{C} \mid (\nu a)\mathcal{C}$$

Los contextos de procesos buscan identificar un “unguarded hole”, es decir un agujero que no tiene prefijada una acción sobre un mailbox.

Def 4: Mailbox conformant

P es *mailbox conformant* si $P \not\rightarrow^* \mathcal{C}[\text{fail } a]$ para todo \mathcal{C} y a .
En el ejemplo del lock, ser *mailbox conformant* significa nunca liberar el lock antes de adquirirlo.

Mailbox calculus

Caracterizaciones operacionales

Def 5: Deadlock free

P es *deadlock free* si $P \rightarrow^* Q \not\vdash$ implica $Q \equiv \text{done}$.

Un proceso se considera *deadlock free* si al terminar tenemos que (1) no hay subprocesos esperando un mensaje que nunca se va a producir y (2) todos los mailbox están vacíos.

Def 7: Fairly terminating

P es *fairly terminating* si $P \rightarrow^* Q$ implica que $Q \rightarrow^* \text{done}$.

Es una propiedad más fuerte que deadlock freedom. Si un proceso es *fairly terminating* entonces se garantiza *junk freedom* (no quedan mensajes sin consumir en ningún mailbox).

Mailbox type system

Syntax

Type τ, σ $::=$ $?E$ (input)
 | $!E$ (output)

Pattern E, F $::=$ \emptyset (unreliable mailbox)
 | $\mathbb{1}$ (empty mailbox)
 | $\mathbf{m}[\overline{\tau}]$ (atom)
 | $E + F$ (sum)
 | $E \cdot F$ (product)
 | E^* (exponential)

Mailbox type system

Patrones

Los patrones son *expresiones regulares conmutativas* que describen las configuraciones válidas de los mensajes dentro de un mailbox.

- ▶ \emptyset : *unreliable mailbox* que recibió un mensaje inesperado.
- ▶ $A + B$: contiene un mensaje A o un mensaje B pero no ambos.
- ▶ $A + \mathbb{1}$: contiene un mensaje A o está vacío.
- ▶ $A \cdot B$: contiene un mensaje A y un mensaje B .
- ▶ A^* : contiene un cantidad arbitraria (incluso 0) de mensajes A .

Mailbox type system

Capabilities

Un *mailbox type* consiste en un *capability* (? o !) junto a un patrón. Un proceso debe cumplir ciertas obligaciones y tiene ciertas garantías descritas por el mailbox type asociado al mailbox que usa.

- ▶ !A: el proceso **debe** escribir un mensaje A en el mailbox.
- ▶ ?A: el proceso tiene **garantizado** recibir un mensaje A.

Mailbox type system

Capabilities: más ejemplos

- ▶ $!(A + \mathbb{1})$: el proceso **puede** escribir un mensaje A en el mailbox, pero no está obligado a hacerlo.
- ▶ $!(A + B)$: el proceso **debe** escribir un mensaje A o B , pero puede **elegir** cuál.
- ▶ $?(A + B)$: el proceso **debe** estar preparado para recibir tanto un mensaje A como B .
- ▶ $?(A \cdot B)$: el proceso tiene **garantizado** recibir un mensaje A y otro B , y puede elegir en qué orden recibirlos.
- ▶ $!(A \cdot B)$: el proceso **debe** escribir un mensaje A y otro B .
- ▶ $!A^*$: el proceso **elige** cuántos mensajes A escribir.
- ▶ $?A^*$: el proceso **debe** estar preparado para recibir una cantidad arbitraria de mensajes A .

Mailbox type system

Semántica de patrones

La semántica de los patrones se define como conjuntos de multiconjuntos de átomos: $\mathbf{m}[\bar{\tau}]$.

$$\begin{array}{lll} \llbracket 0 \rrbracket \stackrel{\text{def}}{=} \emptyset & \llbracket E + F \rrbracket \stackrel{\text{def}}{=} \llbracket E \rrbracket \cup \llbracket F \rrbracket & \llbracket \mathbf{M} \rrbracket \stackrel{\text{def}}{=} \{\langle \mathbf{M} \rangle\} \\ \llbracket 1 \rrbracket \stackrel{\text{def}}{=} \{\langle \rangle\} & \llbracket E \cdot F \rrbracket \stackrel{\text{def}}{=} \{A \uplus B \mid A \in \llbracket E \rrbracket, B \in \llbracket F \rrbracket\} & \llbracket E^* \rrbracket \stackrel{\text{def}}{=} \llbracket 1 \rrbracket \cup \llbracket E \rrbracket \cup \llbracket E \cdot E \rrbracket \cup \dots \end{array}$$

Dada una relación preorder \mathcal{R} sobre los tipos básicos, escribimos $E \sqsubseteq_{\mathcal{R}} F$ para decir que E es un subpatrón de F si $\langle \mathbf{m}_i[\bar{\tau}_i] \rangle_{i \in I} \in \llbracket E \rrbracket$ implica $\langle \mathbf{m}_i[\bar{\sigma}_i] \rangle_{i \in I} \in \llbracket F \rrbracket$ y además $\bar{\tau}_i \mathcal{R} \bar{\sigma}_i$ para todo $i \in I$.

Escribimos $\simeq_{\mathcal{R}}$ para denotar $\sqsubseteq_{\mathcal{R}} \cap \sqsupseteq_{\mathcal{R}}$.

Notemos que $\sqsubseteq_{\mathcal{R}}$ es covariante respecto a \mathcal{R} , pues $\bar{\tau} \mathcal{R} \bar{\sigma}$ implica $\mathbf{m}[\bar{\tau}] \sqsubseteq_{\mathcal{R}} \mathbf{m}[\bar{\sigma}]$.

Mailbox type system

Subtipado

Decimos que \mathcal{R} es una *relación de subtipado* si $\tau \mathcal{R} \sigma$ implica

1. $\tau = ?E$ y $\sigma = ?F$ y $E \sqsubseteq_{\mathcal{R}} F$, o bien
2. $\tau = !E$ y $\sigma = !F$ y $F \sqsubseteq_{\mathcal{R}} E$

Escribimos \leq para denotar la mayor relación de subtipado y decimos que τ es un subtipo de σ si $\tau \leq \sigma$.

Escribimos \leqslant para $\leq \cap \geq$, \sqsubseteq para \sqsubseteq_{\leq} y \simeq para \simeq_{\leq} .

Las 2 reglas se corresponden directamente con las reglas covariantes y contravariantes usuales para canales con capacidades de entrada y salida.

Mailbox type system

Subtipado: ejemplos

- ▶ $?A \leqslant ?(A + B)$: un mailbox de tipo $?A$ ofrece garantías más fuertes que otro de tipo $?(A + B)$. Si un proceso sabe usar un mailbox donde pueden haber mensajes A o B , también sabe usar un mailbox donde solo hay mensajes A .
- ▶ $!(A + B) \leqslant !A$: un mailbox de tipo $!(A + B)$ es más permisivo que otro de tipo $!A$. Si un proceso necesita un mailbox para escribir un mensaje A , también le sirve un mailbox donde se puede escribir mensajes A o B .

Mailbox type system

Relaciones de subtipado especiales

Algunos mailbox types tienen patrones que están en cierta relación particular con las constantes $\mathbb{0}$ (unreliable mailbox) y $\mathbb{1}$ (empty mailbox). Estos tipos tienen la siguiente clasificación:

- ▶ **relevant**: si $\tau \not\leq \mathbb{1}$ (caso contrario *irrelevant*)
Un mailbox *relevant* debe usarse, mientras que uno *irrelevant* puede descartarse. Todos los mailbox con input capability son *relevantes*.
- ▶ **reliable**: si $\tau \not\leq ?\mathbb{0}$ (caso contrario *unreliable*)
Un mailbox *reliable* no recibió mensajes inesperados. Todos los mailbox con output capability son *reliable*.
- ▶ **usable**: si $!\mathbb{0} \not\leq \tau$ (caso contrario *unusable*)
Un mailbox *usable* puede ser usado. Todos los mailbox con input capability son *usable*.

Mailbox type system

Ejemplo 11: lock type

$$\begin{aligned}\text{FreeLock}(self) &\triangleq \text{free } self.\text{done} \\ &\quad + self?\text{acquire}(\text{owner}).\text{BusyLock}[self, \text{owner}] \\ &\quad + self?\text{release}.\text{fail } self \\ \text{BusyLock}(self, \text{owner}) &\triangleq \text{owner}!\text{reply}[self] \mid self?\text{release}.\text{FreeLock}[self]\end{aligned}$$

El mailbox usado por FreeLock tendrá diferentes tipos dependiendo del estado interno del lock y desde qué óptica lo miramos.

- ▶ Desde FreeLock: $?\text{acquire}[\text{!reply}[\text{!release}]]^*$
- ▶ Desde BusyLock: $!(\text{release} \cdot \text{acquire}[\text{!reply}[\text{!release}]]^*)$
- ▶ Desde User hacia FreeLock: $!\text{acquire}[\text{!reply}[\text{!release}]]^*$
- ▶ Desde Owner hacia BusyLock: $!\text{release}$

Mailbox type system

Grafo de dependencias: *sintaxis*

Dependency Graph $\varphi, \psi ::= \emptyset \mid \{u, v\} \mid \varphi \sqcup \psi \mid (\nu a)\varphi$

El grafo de dependencias es un multigrafo no dirigido donde los vértices del grafo son los nombres de los mailbox y el **objetivo es trackear las dependencias entre mailboxes**. Intuitivamente hay una dependencia entre u y v si:

- ▶ v es un argumento de algún mensaje del mailbox u ,
- ▶ o bien v aparece en la continuación de un proceso esperando un mensaje en u .

Mailbox type system

Grafo de dependencias: LTS

$$\begin{array}{c} \{u, v\} \xrightarrow{u-v} \emptyset \quad [\text{G-AXIOM}] \qquad \frac{\varphi \xrightarrow{u-v} \varphi'}{\varphi \sqcup \psi \xrightarrow{u-v} \varphi' \sqcup \psi} \quad [\text{G-LEFT}] \qquad \frac{\psi \xrightarrow{u-v} \psi'}{\varphi \sqcup \psi \xrightarrow{u-v} \varphi \sqcup \psi'} \quad [\text{G-RIGHT}] \\[2ex] \frac{\varphi \xrightarrow{u-v} \psi \quad a \neq u, v}{(\nu a)\varphi \xrightarrow{u-v} (\nu a)\psi} \quad [\text{G-NEW}] \qquad \frac{\varphi \xrightarrow{u-w} \psi \quad \psi \xrightarrow{w-v} \varphi'}{\varphi \xrightarrow{u-v} \varphi'} \quad [\text{G-TRANS}] \end{array}$$

La semántica del grafo de dependencias está dada por un LTS donde el label $u - v$ representa un camino que conecta u con v . La relación $\varphi \xrightarrow{u-v} \varphi'$ describe que u está conectado con v en φ , y φ' describe el grafo residual luego de eliminar las aristas usadas para conectar u con v .

Mailbox type system

Grafo de dependencias: propiedades

Def 12: graph acyclicity and entailment

Sea $\text{dep}(\varphi) \stackrel{\text{def}}{=} \{(u, v) \mid \exists \varphi' : \varphi \xrightarrow{u-v} \varphi'\}$ la relación de dependencias generada por φ .

- ▶ Decimos que φ es *acíclico* si $\text{dep}(\varphi)$ es irreflexiva.
- ▶ Decimos que φ *entails* (implica) ψ , escrito $\varphi \Rightarrow \psi$, si $\text{dep}(\psi) \subseteq \text{dep}(\varphi)$.

Ejemplo 2: Future variable

Definición

$$\begin{aligned}\text{Future}(self) &\triangleq self?\text{put}(x).\text{Present}[self, x] \\ \text{Present}(self, x) &\triangleq \text{free } self.\text{done} \\ &\quad + self?\text{get}(sender).(sender!\text{reply}[x] \mid \text{Present}[self, x]) \\ &\quad + self?\text{put}.\text{fail } self\end{aligned}$$

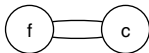
- ▶ Una variable futura admite un único mensaje **put** para setear el valor de la variable.
- ▶ Puede recibir una cantidad arbitraria de mensajes **get**, antes y después de ser seteada. Si se reciben antes del **put**, estos mensajes quedan pendientes en el mailbox ya que el proceso definido por Future no tiene una guarda que selecciona los mensajes **get**.

Ejemplo 2: Future variable

Detección de deadlock con el grafo de dependencias

$$(\nu f)(\nu c)(\text{Future}[f] \mid f!\text{get}[c] \mid c?\text{reply}(x).\text{free } c.f!\text{put}[x])$$

- El mailbox c es el argumento del mensaje **get** guardado en el mailbox f .
- El mailbox f aparece en la continuación luego de leer del mailbox c .



Claramente el grafo de dependencias **tiene un ciclo**. Vamos a utilizar estos grafos en las reglas de tipado para evitar deadlocks.

Mailbox type system

Reglas de tipado

Type environments

Γ y Δ son funciones parciales de nombres a tipos: $\bar{u} : \bar{\tau}$.

Procesos

$\Gamma \vdash P :: \varphi$

P está bien tipado bajo Γ y genera el grafo de dependencias φ .

Decimos que el juicio de tipado está bien formado si

$\text{fn}(\varphi) \subseteq \text{dom}(\Gamma)$ y φ es acíclico.

Guardas

$\Gamma \vdash G$

Mailbox type system

Reglas de tipado

Typing rules for processes

$$\boxed{\Gamma \vdash P :: \varphi}$$

$$\begin{array}{c} \frac{}{\emptyset \vdash \text{done} :: \emptyset} \text{[T-DONE]} \quad \frac{X : (\bar{x} : \bar{\tau}; \varphi)}{\bar{u} : \bar{\tau} \vdash X[\bar{u}] :: \varphi\{\bar{u}/\bar{x}\}} \text{[T-DEF]} \quad \frac{\Gamma, a : ?\mathbb{1} \vdash P :: \varphi}{\Gamma \vdash (\nu a)P :: (\nu a)\varphi} \text{[T-NEW]} \\[10pt] \frac{}{u : !\mathbf{m}[\bar{\tau}], \bar{v} : \bar{\tau} \vdash u!\mathbf{m}[\bar{v}] :: \{u, \{\bar{v}\}\}} \text{[T-MSG]} \quad \frac{u : ?E, \Gamma \vdash G \quad \models E}{u : ?E, \Gamma \vdash G :: \{u, \text{dom}(\Gamma)\}} \text{[T-GUARD]} \\[10pt] \frac{\Gamma_i \vdash P_i :: \varphi_i \quad (i=1,2)}{\Gamma_1 \parallel \Gamma_2 \vdash P_1 \mid P_2 :: \varphi_1 \sqcup \varphi_2} \text{[T-PAR]} \quad \frac{\Delta \vdash P :: \psi \quad \Gamma \leq \Delta \quad \varphi \Rightarrow \psi}{\Gamma \vdash P :: \varphi} \text{[T-SUB]} \end{array}$$

Typing rules for guards

$$\boxed{\Gamma \vdash G}$$

$$\begin{array}{c} \frac{}{u : ?\emptyset, \Gamma \vdash \text{fail} \ u} \text{[T-FAIL]} \quad \frac{\Gamma \vdash P :: \varphi}{u : ?\mathbb{1}, \Gamma \vdash \text{free} \ u.P} \text{[T-FREE]} \\[10pt] \frac{u : ?E, \Gamma, \bar{x} : \bar{\tau} \vdash P :: \varphi}{u : ?(\mathbf{m}[\bar{\tau}] \cdot E), \Gamma \vdash u?\mathbf{m}(\bar{x}).P} \text{[T-IN]} \quad \frac{u : ?E_i, \Gamma \vdash G_i \quad (i=1,2)}{u : ?(E_1 + E_2), \Gamma \vdash G_1 + G_2} \text{[T-BRANCH]} \end{array}$$

Mailbox type system

[T-MSG]

$$\frac{}{u : !\mathbf{m}[\bar{\tau}], \bar{v} : \bar{\tau} \vdash u!\mathbf{m}[\bar{v}] :: \{u, \{\bar{v}\}\}} \quad [\text{T-MSG}]$$

Un mensaje está bien tipado si el mailbox u admite recibir un mensaje con tag \mathbf{m} y argumentos de tipo $\bar{\tau}$, y además los valores de los argumentos \bar{v} efectivamente tienen tipo $\bar{\tau}$.

Enviar un mensaje genera una dependencia entre el mailbox u y todos los argumentos \bar{v} .

$$\{u, \{\bar{v}\}\} \equiv \{u, v_1\} \sqcup \cdots \sqcup \{u, v_n\}$$

Mailbox type system

[T-FAIL] y [T-FREE]

$$\frac{}{u : ?\mathbb{0}, \Gamma \vdash \text{fail } u} \quad [\text{T-FAIL}] \qquad \frac{\Gamma \vdash P :: \varphi}{u : ?\mathbb{1}, \Gamma \vdash \text{free } u.P} \quad [\text{T-FREE}]$$

La regla [T-FAIL] permite tipar un *runtime error* si el mailbox u tiene tipo $?\mathbb{0}$ indicando que el mailbox es *unreliable* (recibió un mensaje inesperado).

La regla [T-FREE] dice que podemos liberar el mailbox u si tiene tipo $?\mathbb{1}$ indicando que está vacío, siempre y cuando la continuación P está bien tipada en el ambiente residual.

Mailbox type system

[T-IN]

$$\frac{u : ?E, \Gamma, \bar{x} : \bar{\tau} \vdash P :: \varphi}{u : ?(\mathbf{m}[\bar{\tau}] \cdot E), \Gamma \vdash u?_{\mathbf{m}}(\bar{x}).P} \quad [\text{T-IN}]$$

Consumir un mensaje \mathbf{m} de un mailbox u requiere que el mailbox tenga tipo $?(\mathbf{m}[\bar{\tau}] \cdot E)$ que garantiza que hay al menos 1 mensaje \mathbf{m} , posiblemente con otros mensajes acorde al patrón E .

La continuación P debe estar bien tipada en un ambiente donde el mailbox tiene tipo $?E$ que describe el contenido del mailbox luego de haber consumido el mensaje \mathbf{m} , y además incluye el tipo de los argumentos \bar{x} del mensaje.

Mailbox type system

[T-BRANCH]

$$\frac{u : ?E_i, \Gamma \vdash G_i \quad (i=1,2)}{u : ?(E_1 + E_2), \Gamma \vdash G_1 + G_2} \quad [\text{T-BRANCH}]$$

La composición de guardas $G_1 + G_2$ ofrece las acciones de G_1 y G_2 donde cada una se corresponde con el subpatrón E_1 y E_2 del mailbox type $?(E_1 + E_2)$.

Mailbox type system

[T-GUARD]

$$\frac{u : ?E, \Gamma \vdash G \quad \models E}{u : ?E, \Gamma \vdash G :: \{u, \text{dom}(\Gamma)\}} \quad [\text{T-GUARD}]$$

Esta regla tipa un *guarded process* G que posiblemente consume mensajes del mailbox u . El grafo de dependencias se define entre u y todas las variables libres que aparecen en la continuación.

La condición $\models E$ pide que el patrón E esté en **forma normal**.

Mailbox type system

Patrones en forma normal

$$u : ?(\mathbf{A} \cdot \mathbf{C} + \mathbf{B} \cdot \mathbf{A}) \vdash u ? \mathbf{A}.P + u ? \mathbf{B}.Q$$

Si usamos la regla [T-IN] para tipar ambas guardas, necesitamos tipar P y Q en un ambiente donde el mailbox u fue actualizado para reflejar que se consumió un mensaje. Podríamos inferir que para P tenemos $u : ?\mathbf{C}$ y para Q tenemos $u : ?\mathbf{A}$.

Esto está mal porque el tipo de u especifica que un mensaje \mathbf{A} puede estar acompañado de otro mensaje \mathbf{C} o \mathbf{B} . La solución es llevar los patrones a una **forma normal** para evitar estos problemas. En este ejemplo, una forma normal podría ser:
 $u : ?\mathbf{A} \cdot (\mathbf{B} + \mathbf{C})$.

Mailbox type system

Propiedades

Teorema 23

Teorema 24

Teorema 25

Encoding binary sessions

Related work

Conclusiones