Mailbox Types for Unordered Interactions[1]

Presentación por Jonathan Bekenstein

Materia optativa sobre Tipos Comportamentales y Contratos

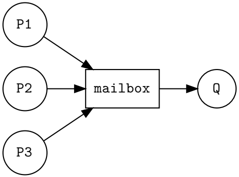
2024

[1] Ugo 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 los procesos pueden:

- escribir mensajes identificados con un *tag* y argumentos,
- consumir (leer) 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

$$
\begin{array}{rrcll}
\textbf{Process} & P, Q & ::= & \texttt{done} & \text{(termination)} \\
 & & | & \mathsf{X}[\overline{u}] & \text{(invocation)} \\
 & & | & G & \text{(guarded process)} \\
 & & | & u\,\texttt{!}\,\texttt{m}[\overline{v}] & \text{(stored message)} \\
 & & | & P \mid Q & \text{(parallel composition)} \\
 & & | & (\nu a)P & \text{(mailbox restriction)} \\
 & & & & \\
\textbf{Guard} & G, H & ::= & \texttt{fail}\ u & \text{(runtime error)} \\
 & & | & \texttt{free}\ u\,.\,P & \text{(mailbox deletion)} \\
 & & | & u\,\texttt{?}\,\texttt{m}(\overline{x})\,.\,P & \text{(selective receive)} \\
 & & | & G + H & \text{(guard composition)}
\end{array}
$$

$\overline{u}$ denota la secuencia $u_1, \ldots, u_n$

Mailbox calculus

Sintaxis: mensajes

Enviar mensajes

$u!\mathtt{m}[\overline{v}]$

Guarda un mensaje identificado con el tag $\mathtt{m}$ y argumentos $\overline{v}$ en el mailbox u.

Recibir mensajes

$u?\mathtt{m}(\overline{x}).P$

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

Mailbox calculus

Sintaxis: procesos

Invocación

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

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

$$
\begin{array}{lrll}
[\text{R-READ}] & a\,\mathtt{!m}[\overline{c}] \mid a\mathtt{?m}(\overline{x})\,.\,P + G & \rightarrow P\{\overline{c}/\overline{x}\} & \\
[\text{R-FREE}] & (\nu a)(\mathtt{free}\ a\,.\,P + G) & \rightarrow P & \\
[\text{R-DEF}] & \mathsf{X}[\overline{c}] & \rightarrow P\{\overline{c}/\overline{x}\} & \text{if } \mathsf{X}(\overline{x}) \triangleq P \\
[\text{R-PAR}] & P \mid R & \rightarrow Q \mid R & \text{if } P \rightarrow Q \\
[\text{R-NEW}] & (\nu a)P & \rightarrow (\nu a)Q & \text{if } P \rightarrow Q \\
[\text{R-STRUCT}] & P & \rightarrow Q & \text{if } P \equiv P' \rightarrow Q' \equiv Q
\end{array}
$$

Relación de congruencia estructural

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

Ejemplo 1: Lock

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

$$\mathsf{User}(\mathit{self}, \mathit{lock}) \triangleq \mathit{lock}!\texttt{acquire}[\mathit{self}] \mid \mathit{self}?\texttt{reply}(l).(l!\texttt{release} \mid \texttt{free}\ \mathit{self}\,.\texttt{done})$$

$$(\nu\, \mathit{lock})(\nu\, \mathit{alice})(\nu\, \mathit{carol})(\mathsf{FreeLock}[\mathit{lock}] \mid \mathsf{User}[\mathit{alice}, \mathit{lock}] \mid \mathsf{User}[\mathit{carol}, \mathit{lock}])$$

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 capabilidad de procesar este mensaje.

Mailbox type system

Sintaxis

$$
\begin{array}{llcll}
\textbf{Type} & \tau, \sigma & ::= & \mathtt{?}E & \text{(input)} \\
 & & | & \mathtt{!}E & \text{(output)} \\
 & & & & \\
\textbf{Pattern} & E, F & ::= & \mathbb{0} & \text{(unreliable mailbox)} \\
 & & | & \mathbb{1} & \text{(empty mailbox)} \\
 & & | & \mathtt{m}[\overline{\tau}] & \text{(atom)} \\
 & & | & E + F & \text{(sum)} \\
 & & | & E \cdot F & \text{(product)} \\
 & & | & E^{*} & \text{(exponential)}
\end{array}
$$

Mailbox type system

Patrones

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

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

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 descriptas 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

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

Mailbox type system

Subtipado

Escribimos $\leqslant$ para denotar la mayor relación de subtipado.

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

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

Mailbox type system

Ejemplo 11: lock type

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

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

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

Mailbox type system

[T-MSG]

$$\frac{}{u : \mathtt{!m}[\overline{\tau}], \overline{v} : \overline{\tau} \vdash u\mathtt{!m}[\overline{v}] :: \{u, \{\overline{v}\}\}} \;\; \text{[T-MSG]}$$

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

Enviar un mensaje genera una dependencia entre el mailbox u y todos los argumentos $\overline{v}$. **¿Cómo evitamos un deadlock?**

Grafo de dependencias

El grafo de dependencias φ 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

[T-FAIL] y [T-FREE]

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

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]

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

Consumir un mensaje $\mathtt{m}$ de un mailbox u requiere que el mailbox tenga tipo $?(\mathtt{m}[\overline{\tau}] \cdot E)$ que garantiza que hay al menos 1 mensaje $\mathtt{m}$, 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 $\mathtt{m}$, y además incluye el tipo de los argumentos $\overline{x}$ del mensaje.

Mailbox type system

[T-BRANCH]

$$\frac{u : \mathtt{?}E_i, \Gamma \vdash G_i \;^{(i=1,2)}}{u : \mathtt{?}(E_1 + E_2), \Gamma \vdash G_1 + G_2} \;[\textsc{t-branch}]$$

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]

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

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

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

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 $u :?\mathtt{C}$ y para Q tenemos $u :?\mathtt{A}$.

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

Mailbox type system

[T-PAR]

$$\frac{\Gamma_i \vdash P_i :: \varphi_i \;\; {}^{(i=1,2)}}{\Gamma_1 \parallel \Gamma_2 \vdash P_1 \mid P_2 :: \varphi_1 \sqcup \varphi_2} \;\; \text{[T-PAR]}$$

La composición paralela $P_1|P_2$ posiblemente necesite tipar 2 usos distintos del mismo mailbox u, ya que P_1 podría escribir un mensaje `A` en u mientras que P_2 podría escribir otro mensaje `B`. En la composición paralela $P_1|P_2$ necesitamos tipar u con un único tipo que contemple ambos mensajes.

Mailbox type system

Combinación de tipos

$$!E \parallel !F \stackrel{\text{def}}{=} !(E \cdot F) \qquad !E \parallel ?(E \cdot F) \stackrel{\text{def}}{=} ?F \qquad ?(E \cdot F) \parallel !E \stackrel{\text{def}}{=} ?F$$

- $!\mathtt{A} \parallel !\mathtt{B} = !(\mathtt{A} \cdot \mathtt{B})$
  Guardar un mensaje $\mathtt{A}$ y otro $\mathtt{B}$ es equivalente al patrón producto $\mathtt{A} \cdot \mathtt{B}$.
- $!\mathtt{A} \parallel ?(\mathtt{A} \cdot \mathtt{B}) = ?\mathtt{B}$
  Cuando el mailbox se usa para input y output, el tipo combinado es el **balance final de mensajes** en el mailbox.

El operador $\parallel$ se extiende inductivamente para la combinación de ambientes: $\Gamma \parallel \Delta$.

Mailbox type system

[T-NEW]

$$\frac{\Gamma, a : \mathtt{?}\mathbb{1} \vdash P :: \varphi}{\Gamma \vdash (\nu a)P :: (\nu a)\varphi} \text{ [T-NEW]}$$

Al crear un nuevo mailbox a con scope P pedimos que el tipo de a sea $?\mathbb{1}$ (mailbox vacío) indicando que todos los mensajes producidos por los subprocesos de P son todos eventualmente consumidos (también por subprocesos de P).

Mailbox type system

Reglas de tipado

Typing rules for processes $\boxed{\Gamma \vdash P :: \varphi}$

$$\frac{}{\emptyset \vdash \texttt{done} :: \emptyset}\ [\text{T-DONE}] \qquad \frac{\mathsf{X} : (\overline{x} : \overline{\tau}; \varphi)}{\overline{u} : \overline{\tau} \vdash \mathsf{X}[\overline{u}] :: \varphi\{\overline{u}/\overline{x}\}}\ [\text{T-DEF}] \qquad \frac{\Gamma, a : \texttt{?}\mathbb{1} \vdash P :: \varphi}{\Gamma \vdash (\nu a)P :: (\nu a)\varphi}\ [\text{T-NEW}]$$

$$\frac{}{u : \texttt{!}\texttt{m}[\overline{\tau}], \overline{v} : \overline{\tau} \vdash u\texttt{!}\texttt{m}[\overline{v}] :: \{u, \{\overline{v}\}\}}\ [\text{T-MSG}] \qquad \frac{u : \texttt{?}E, \Gamma \vdash G \qquad \models E}{u : \texttt{?}E, \Gamma \vdash G :: \{u, \mathsf{dom}(\Gamma)\}}\ [\text{T-GUARD}]$$

$$\frac{\Gamma_i \vdash P_i :: \varphi_i \ ^{(i=1,2)}}{\Gamma_1 \parallel \Gamma_2 \vdash P_1 \mid P_2 :: \varphi_1 \sqcup \varphi_2}\ [\text{T-PAR}] \qquad \frac{\Delta \vdash P :: \psi \qquad \Gamma \leqslant \Delta \qquad \varphi \Rightarrow \psi}{\Gamma \vdash P :: \varphi}\ [\text{T-SUB}]$$

Typing rules for guards $\boxed{\Gamma \vdash G}$

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

$$\frac{u : \texttt{?}E, \Gamma, \overline{x} : \overline{\tau} \vdash P :: \varphi}{u : \texttt{?}(\texttt{m}[\overline{\tau}] \cdot E), \Gamma \vdash u\texttt{?}\texttt{m}(\overline{x})\,.\,P}\ [\text{T-IN}] \qquad \frac{u : \texttt{?}E_i, \Gamma \vdash G_i \ ^{(i=1,2)}}{u : \texttt{?}(E_1 + E_2), \Gamma \vdash G_1 + G_2}\ [\text{T-BRANCH}]$$

Mailbox type system

Juicios bien formados

$\Gamma \vdash P :: \varphi$

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

Decimos que el juicio de tipado está bien formado si
$\mathsf{fn}(\varphi) \subseteq \mathsf{dom}(\Gamma)$ y φ es acíclico.

Intuición

$\Gamma =$ **mensaje producidos por P - mensajes consumidos por P.**

Def 4: Mailbox conformant

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

Def 5: Deadlock free

P es *deadlock free* si $P \rightarrow^* Q \not\rightarrow$ implica $Q \equiv \texttt{done}$.
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 $Q \rightarrow^*$ `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).

Propiedades de procesos bien tipados

Teo 23: Subject reduction (preservación de tipos)

Si Γ es *reliable*, $\Gamma \vdash P :: \varphi$ y $P \rightarrow Q$ entonces $\Gamma \vdash Q :: \varphi$

En contraste con otros tipos comportamentales, en particular con session types, los mailbox types usados por un proceso no necesitan realizar una reducción de tipo.

Este resultado se ancla fuertemente en la idea de que el tipo del mailbox refleja el **balance** final de mensajes. La preservación de tipos garantiza que un proceso no va a consumir más mensajes de los que se producen, ni va a producir más mensajes de los que se consumen.

Propiedades de procesos bien tipados

Teo 24: Soundness (resultado principal)

Si $\emptyset \vdash P :: \varphi$ entonces P es *mailbox conformant* y *deadlock free*.

Los procesos cerrados y bien tipados tienen la garantía de ser *mailbox conformant* (no reciben mensajes inesperados) y además son *deadlock free*.

Propiedades de procesos bien tipados

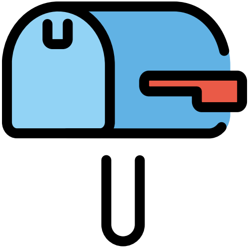
Teo 25: Fair termination

Si $\emptyset \vdash P :: \varphi$ y P es *finitely unfolding*
entonces P es *fairly terminating*.

Decimos que P es *finitely unfolding* si todas las reducciones maximales de P usan `[R-DEF]` (invocación de procesos) una cantidad **finita** de veces.

La intuición es que P es fairly terminating si se invoca una cantidad finita de procesos. Al estar P bien tipado bajo el ambiente vacío, concluimos que todos los mailboxes tienen un balance final nulo, es decir terminan vacíos, y por lo tanto se puede garantizar *junk freedom* (no hay mensajes sin consumir).

Fin



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

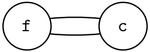
- Una variable futura admite un único mensaje `put` para setear el valor de la variable.
- Puede recibir una cantidad arbitraria de mensajes `get`, antes y después de ser seteada. Si se reciben antes del `put`, estos mensajes quedan pendientes en el mailbox ya que el proceso definido por Future no tiene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

Apéndice

Detección de deadlock con el grafo de dependencias

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

- 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.

Contextos de procesos

$$\mathscr{C} \quad ::= \quad [\,] \quad | \quad \mathscr{C} \mid P \quad | \quad P \mid \mathscr{C} \quad | \quad (\nu a)\mathscr{C}$$

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

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

$$
\begin{array}{lll}
[\![\mathbb{0}]\!] \stackrel{\text{def}}{=} \emptyset & [\![E+F]\!] \stackrel{\text{def}}{=} [\![E]\!] \cup [\![F]\!] & [\![\mathtt{M}]\!] \stackrel{\text{def}}{=} \{\langle \mathtt{M} \rangle\} \\
[\![\mathbb{1}]\!] \stackrel{\text{def}}{=} \{\langle\rangle\} & [\![E \cdot F]\!] \stackrel{\text{def}}{=} \{\mathsf{A} \uplus \mathsf{B} \mid \mathsf{A} \in [\![E]\!], \mathsf{B} \in [\![F]\!]\} & [\![E^*]\!] \stackrel{\text{def}}{=} [\![\mathbb{1}]\!] \cup [\![E]\!] \cup [\![E \cdot E]\!] \cup \cdots
\end{array}
$$

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

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

Notemos que $\sqsubseteq_{\mathscr{R}}$ es covariante respecto a $\mathscr{R}$,
pues $\overline{\tau} \mathrel{\mathscr{R}} \overline{\sigma}$ implica $\mathtt{m}[\overline{\tau}] \sqsubseteq_{\mathscr{R}} \mathtt{m}[\overline{\sigma}]$.

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

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

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

Escribimos $\lesseqgtr$ para $\leqslant \cap \geqslant$, $\sqsubseteq$ para $\sqsubseteq_{\leqslant}$ y $\simeq$ para $\simeq_{\leqslant}$.

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

Apéndice

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

Apéndice

Grafo de dependencias: sintaxis

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

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.

Apéndice

Grafo de dependencias: LTS

$$\{u, v\} \xrightarrow{u-v} \emptyset \quad \text{[G-AXIOM]} \qquad \frac{\varphi \xrightarrow{u-v} \varphi'}{\varphi \sqcup \psi \xrightarrow{u-v} \varphi' \sqcup \psi} \quad \text{[G-LEFT]} \qquad \frac{\psi \xrightarrow{u-v} \psi'}{\varphi \sqcup \psi \xrightarrow{u-v} \varphi \sqcup \psi'} \quad \text{[G-RIGHT]}$$

$$\frac{\varphi \xrightarrow{u-v} \psi \qquad a \neq u, v}{(\nu a)\varphi \xrightarrow{u-v} (\nu a)\psi} \quad \text{[G-NEW]} \qquad \frac{\varphi \xrightarrow{u-w} \psi \qquad \psi \xrightarrow{w-v} \varphi'}{\varphi \xrightarrow{u-v} \varphi'} \quad \text{[G-TRANS]}$$

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.

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 $\text{dep}(\psi) \subseteq \text{dep}(\varphi)$.