Mailbox Types for Unordered Interactions[1]

Presentación por Jonathan Bekenstein

Materia optativa sobre Tipos Comportamentales y Contratos

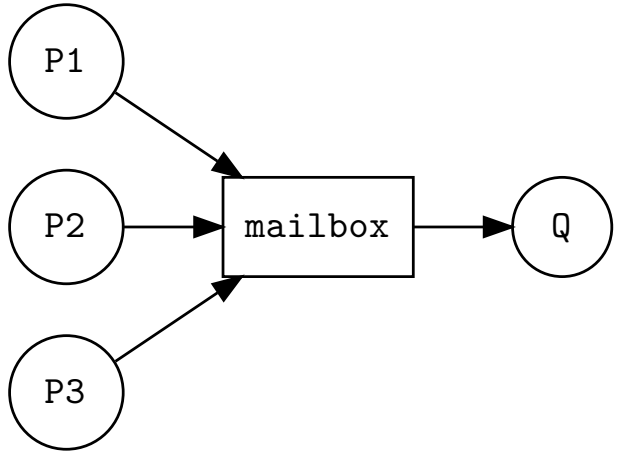
2024

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

Introducción

Modelar interacciones **asincrónicas** sobre diferentes topologías de procesos concurrentes. La comunicación es **multi-party** y sucede mediante el uso de un **mailbox**, 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*).



Mailbox calculus

Sintaxis: gramática

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

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

Mailbox calculus

Sintaxis: mensajes

Enviar mensajes

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

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

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

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

Paralelo y restricción

$P \,|\, Q$ 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

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

Relación de congruencia estructural

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

Ejemplo 1: Lock

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

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

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

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 capabilidad de procesar este mensaje.

Mailbox calculus

Caracterizaciones operacionales

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.

Mailbox calculus

Caracterizaciones operacionales

Def 5: Deadlock free

P es *deadlock free* si $P \to^* Q \not\to$ 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 $P \to^* Q$ implica que $Q \to^* \texttt{done}$.
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).

Ejemplo 2: Future variable

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

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

Es legal hacer un `get` de una variable futura antes de que sea resuelta con `resolve`. Este proceso es *mailbox conformant* pero está en deadlock. Para evitar que este proceso esté bien tipado se usan *dependency graphs*.

Mailbox type system

Sintaxis

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

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

Semántica de patrones

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

$$\llbracket \mathbb{0} \rrbracket \stackrel{\text{def}}{=} \emptyset \qquad \llbracket E + F \rrbracket \stackrel{\text{def}}{=} \llbracket E \rrbracket \cup \llbracket F \rrbracket \qquad \llbracket \mathtt{M} \rrbracket \stackrel{\text{def}}{=} \{\langle \mathtt{M} \rangle\}$$
$$\llbracket \mathbb{1} \rrbracket \stackrel{\text{def}}{=} \{\langle\rangle\} \qquad \llbracket E \cdot F \rrbracket \stackrel{\text{def}}{=} \{\mathsf{A} \uplus \mathsf{B} \mid \mathsf{A} \in \llbracket E \rrbracket, \mathsf{B} \in \llbracket F \rrbracket\} \qquad \llbracket E^* \rrbracket \stackrel{\text{def}}{=} \llbracket \mathbb{1} \rrbracket \cup \llbracket E \rrbracket \cup \llbracket E \cdot E \rrbracket \cup \cdots$$

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 $\langle \mathtt{m}_i[\overline{\tau}_i] \rangle_{i \in I} \in \llbracket E \rrbracket$ implica $\langle \mathtt{m}_i[\overline{\sigma}_i] \rangle_{i \in I} \in \llbracket F \rrbracket$ y además $\overline{\tau}_i \mathscr{R} \overline{\sigma}_i$ 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 $\overline{\tau} \mathscr{R} \overline{\sigma}$ implica $\mathtt{m}[\overline{\tau}] \sqsubseteq_{\mathscr{R}} \mathtt{m}[\overline{\sigma}]$.

Mailbox type system

Subtipado

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

Mailbox type system

Subtipado: ejemplos

- $?\mathtt{A} \leqslant ?(\mathtt{A} + \mathtt{B})$: 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 `A` o `B`, también sabe usar un mailbox donde solo hay mensajes `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 `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 $\tau \not\leqslant !\mathbb{1}$ (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*.

Mailbox type system

Ejemplo 11: lock type

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

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

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

Mailbox type system

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.

Mailbox type system

Grafo de dependencias: LTS

$$\{u, v\} \xrightarrow{u-v} \emptyset \quad \text{[G-AXIOM]}$$

$$\frac{\varphi \xrightarrow{u-v} \varphi'}{\varphi \sqcup \psi \xrightarrow{u-v} \varphi' \sqcup \psi} \quad \text{[G-LEFT]}$$

$$\frac{\psi \xrightarrow{u-v} \psi'}{\varphi \sqcup \psi \xrightarrow{u-v} \varphi \sqcup \psi'} \quad \text{[G-RIGHT]}$$

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

$$\frac{\varphi \xrightarrow{u-w} \psi \qquad \psi \xrightarrow{w-v} \varphi'}{\varphi \xrightarrow{u-v} \varphi'} \quad \text{[G-TRANS]}$$

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.

Mailbox type system

Grafo de dependencias: propiedades

Def 12: graph acyclicity and entailment

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

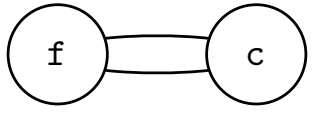
- Decimos que φ es *acíclico* si $\mathsf{dep}(\varphi)$ es irreflexiva.
- Decimos que φ *entails* (implica) ψ, escrito $\varphi \Rightarrow \psi$, si $\mathsf{dep}(\psi) \subseteq \mathsf{dep}(\varphi)$.

Mailbox type system

Grafo de dependencias: ejemplo

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

- 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

Mailbox type system

Propiedades

Teorema 23

Teorema 24

Teorema 25

Encoding binary sessions