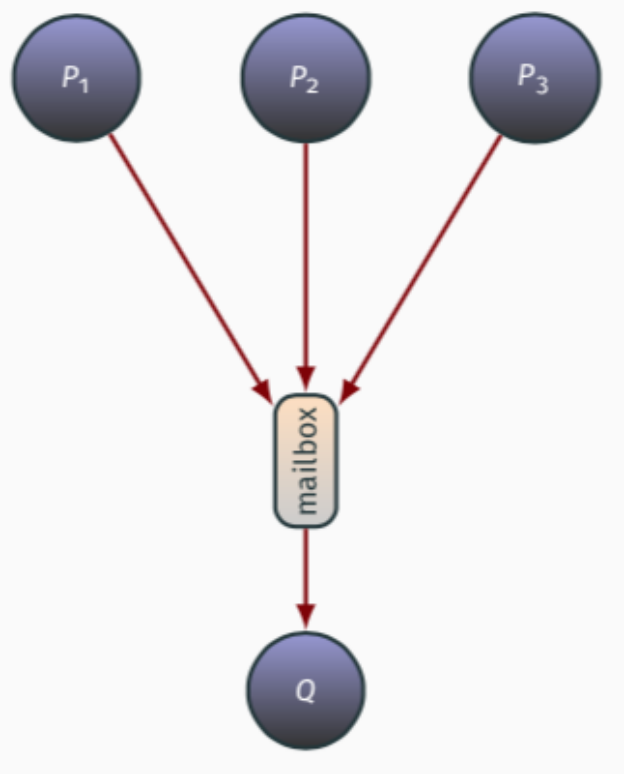
Mailbox Types for Unordered Interactions

Ugo de'Liguoro, Luca Padovani

Università di Torino

TYPES - June 2018, Braga

Static Analysis of Unordered Interactions



A popular communication model...

- **many-to-one** communications
- **selective input**
- used by **actors** (Akka, Erlang, CAF, ...)

...calling for a type system such that

- well-typed processes interact **safely**
- don't receive **unexpected** messages
- don't leave **garbage** behind
- don't **deadlock**

Key Ideas

1. Types describe **mailboxes** (not processes)
2. Subtyping embodies the **unordered** nature of mailboxes
3. Type judgments express **balance** of *obligations* and *expectations*
4. Well-typed processes **break even**

Syntax of Mailbox Calculus - MC

Asynchronous π-calculus + tagged messages + `fail`/`free`

Process	$P, Q ::=$	`done`	(termination)
		$\mathsf{X}[\overline{u}]$	(invocation)
		G	(guard)
		$u!\mathtt{m}[\overline{v}]$	(message)
		$P \mid Q$	(parallel)
		$(\nu a)P$	(mailbox)
Guard	$G, H ::=$	`fail` u	(exception)
		`free` $u\,.\,P$	(deallocation)
		$u?\mathtt{m}(\overline{x})\,.\,P$	(selective input)
		$G + H$	(external choice)

Reduction Semantics

Tags used to **select** received messages

$$a!\mathtt{m}[\overline{c}] \mid a?\mathtt{m}(\overline{x}).P + G \rightarrow P\{\overline{c}/\overline{x}\}$$

Empty mailboxes are explicitly **deallocated**

$$(\nu a)(\mathtt{free}\ a.P + G) \rightarrow P$$

Failure is relevant only when is the only guard:

$$\mathtt{fail}\ a + G \equiv G$$

Syntax of Mailbox Types

$$\begin{array}{rrcl} \text{type} & \tau & ::= & \dagger E \\ \text{capability} & \dagger & ::= & ? \mid ! \\ \text{pattern} & E & ::= & \mathbb{0} \mid \mathbb{1} \mid \mathtt{m}[\overline{\tau}] \mid E + F \mid E \cdot F \mid E^* \end{array}$$

Capabilities

- $?$ = mailbox with **negative** balance (used for **inputs**)
- $!$ = mailbox with **positive** balance (used for **outputs**)

Patterns

- **commutative Kleene algebra** over message types $\mathtt{m}[\overline{\tau}]$
- describe the content of the mailbox

Pattern Semantics

$[\![E]\!]$ = set of *configurations* i.e. multisets of messages $\mathtt{m}[\overline{\tau}]$:

- $[\![\mathbb{0}]\!] = \varnothing$
- $[\![\mathbb{1}]\!] = \text{unit} = \{\langle\rangle\}$, where $\langle\rangle$ is the empty multiset
- $[\![\mathtt{m}[\overline{\tau}]]\!] = \{\langle \mathtt{m}[\overline{\tau}] \rangle\}$
- $[\![E + F]\!] = [\![E]\!] \cup [\![F]\!]$
- $[\![E \cdot F]\!] = [\![E]\!] \uplus [\![F]\!] = \{M \uplus N \mid M \in [\![E]\!], N \in [\![F]\!]\}$
- $[\![E^*]\!] = \text{fix}(\Phi_E) \quad \text{where} \quad \Phi_E(X) = \text{unit} \cup ([\![E]\!] \uplus X)$

Typing Judgments

$$\Gamma \vdash P$$

Intuition

- Γ = messages **produced** by P − messages **consumed** by P

Consequences

- all mailboxes in Γ are **empty** $\iff$ P **breaks even**
- types in Γ are **preserved** by reductions

Type Semantics and Subtyping

$\sigma = !E$ represents an **obligation**:

$u : !E \vdash P$ $\quad$ P may store in u *some* configuration in $[\![E]\!]$

$\tau = ?E$ represents an **expectation**:

$u : ?E \vdash P$ $\quad$ P expects from u *any* configuration in $[\![E]\!]$

Subtyping: let $E \sqsubseteq F$ (roughly) mean $[\![E]\!] \subseteq [\![F]\!]$, then

$$\frac{E \sqsubseteq F}{!F \leqslant !E} \qquad \frac{E \sqsubseteq F}{?E \leqslant ?F} \qquad \frac{u : \tau \vdash P \quad \sigma \leqslant \tau}{u : \sigma \vdash P}$$

Typing Rules: subtyping

Given

$$\frac{}{u : !\mathbb{1}, \Gamma \leqslant \Gamma} \qquad \frac{\sigma \leqslant \tau}{u : \sigma, \Gamma \leqslant u : \tau, \Gamma}$$

from which we have a derived rule of restricted weakening:

$$\frac{\sigma \leqslant !\mathbb{1}}{u : \sigma, \Gamma \leqslant \Gamma}$$

then we have the **subtyping** rule

$$\frac{\Delta \vdash P \qquad \Gamma \leqslant \Delta}{\Gamma \vdash P}$$

Typing Rules: `done`, `fail`

$$\frac{}{\vdash \texttt{done}}$$

$$\frac{}{u : ?\mathbb{0}, \Gamma \vdash \texttt{fail}\ u}$$

Typing Rules: `free` and new

$$\frac{\Gamma \vdash P}{u : ?\mathbb{1}, \Gamma \vdash \texttt{free}\ u\,.\,P}$$

$$\frac{u : ?\mathbb{1}, \Gamma \vdash P}{\Gamma \vdash (\nu u)P}$$

Typing Rules: output and input

$$\frac{}{u : !\mathtt{m}[\overline{T}],\ \overline{v} : \overline{T} \vdash u!\mathtt{m}[\overline{v}]}$$

$$\frac{u : ?E,\ \overline{x} : \overline{T},\ \Gamma \vdash P}{u : ?(\mathtt{m}[\overline{T}] \cdot E),\ \Gamma \vdash u?\mathtt{m}(\overline{x}).P}$$

Typing Rules: branching

$$\frac{u : ?E, \Gamma \vdash G \quad u : ?F, \Gamma \vdash H}{u : ?(E + F), \Gamma \vdash G + H}$$

with side condition that:

$E + F$ is a sum of $\mathbb{0}$, $\mathbb{1}$ or of patterns $\mathtt{M} \cdot E'$ (normal form)

Typing Rules: parallel composition

$$\frac{u : !E \vdash P \qquad u : !F \vdash Q}{u : !(E \cdot F) \vdash P \mid Q}$$

$$\frac{u : !E \vdash P \qquad u : ?(E \cdot F) \vdash Q}{u : ?F \vdash P \mid Q}$$

but parallel of input actions is **undefined**

In general

$$\frac{\Gamma \vdash P \qquad \Delta \vdash Q}{\Gamma \| \Delta \vdash P \mid Q}$$

where $\Gamma \| \Delta$, acting pointwise on name typings, must be defined

Example

$$\dfrac{u : !\mathtt{m} \vdash u!\mathtt{m} \qquad \dfrac{\dfrac{\vdash \mathtt{done}}{u : ?\mathbb{1} \vdash \mathtt{free}\ u.\mathtt{done}}}{u : ?(\mathtt{m} \cdot \mathbb{1}) \vdash u?\mathtt{m}.\mathtt{free}\ u.\mathtt{done}}}{u : ?\mathbb{1} \vdash u!\mathtt{m} \mid (u?\mathtt{m}.\mathtt{free}\ u.\mathtt{done})}$$

Now

$$u!\mathtt{m} \mid (u?\mathtt{m}.\mathtt{free}\ u.\mathtt{done}) \rightarrow \mathtt{free}\ u.\mathtt{done} \not\rightarrow$$

and clearly $u : ?\mathbb{1} \vdash \mathtt{free}\ u.\mathtt{done}$ is derivable

Example (cont.)

Actually

$$(\nu u)(u!\mathtt{m} \mid (u?\mathtt{m}.\mathtt{free}\ u.\mathtt{done})) \rightarrow (\nu u)(\mathtt{free}\ u.\mathtt{done}) \rightarrow \mathtt{done}$$

in which case by rule [new] we have

$$\frac{u : ?\mathbb{1} \vdash u!\mathtt{m} \mid (u?\mathtt{m}.\mathtt{free}\ u.\mathtt{done})}{\vdash (\nu u)(u!\mathtt{m} \mid (u?\mathtt{m}.\mathtt{free}\ u.\mathtt{done}))}$$

and both $\vdash (\nu u)(\mathtt{free}\ u.\mathtt{done})$ and $\vdash \mathtt{done}$ are derivable

Subject Reduction and Conformance

Theorem

If $\Gamma \vdash P$ *and* $P \to Q$, *then* $\Gamma \vdash Q$

With other behavioral type systems (e.g. session types) you just have

$$\text{if} \quad \Gamma \vdash P \quad \text{and} \quad P \to Q \quad \text{then} \quad \Delta \vdash Q \quad \text{for some } \Delta$$

A closed P is **conformant** if $P \twoheadrightarrow^* Q \equiv (\nu u)(\texttt{fail}\ u \mid \cdots)$

Corollary

If $\vdash P$ *then* P *is conformant*

Deadlock

The term

$$(\nu u)(\nu v)(u?\mathtt{m}.\mathtt{free}\ u.v!\mathtt{m} \mid v?\mathtt{m}.\mathtt{free}\ v.u!\mathtt{m}) \nrightarrow$$

is typable and hence conformant, but **deadlocked**

Definition (mailbox dependency)

There is a **dependency** between mailboxes u and v if either

- v occurs in the continuation of a process blocked on u
- v occurs in a message stored in u

Dependency Graphs

Dependency graphs are unordered graphs plus local names:

$$\varphi ::= \varnothing \mid \{u, v\} \mid \varphi \sqcup \varphi \mid (\nu u)\varphi$$

Rules are extended to derive judgments

$$\Gamma \vdash P :: \varphi \quad \text{where } \varphi \text{ is acyclic}$$

Theorem

If $\Gamma \vdash P :: \varphi$ then P is conformant and deadlock free

Conclusion and research directions

Achievements:

- a notion of type inspired to regular language theory
- a simple type assignment system for an asynchronous π-calculus where types are invariant properties of names (mailboxes)
- the system is decidable and there is a type reconstruction algorithm (actually we have a tool)
- the system ensures safety and (some) liveness properties

Future work:

- find a mathematical model of types and typing judgments
- relate the system to logic (e.g. linear logic) if it makes sense
- scale the type system to concrete programming languages and libraries, like Java and Scala Akka library

References

- Paper: U. de'Liguoro, L. Padovani, *Mailbox Types for Unordered Interactions.* CoRR abs/1801.04167 (2018) - accepted for ECOOP'18

- Tool: L. Padovani, *MC*^{2} - *Mailbox Calculus Checker*, `http://www.di.unito.it/~padovani/Software/MCC/`