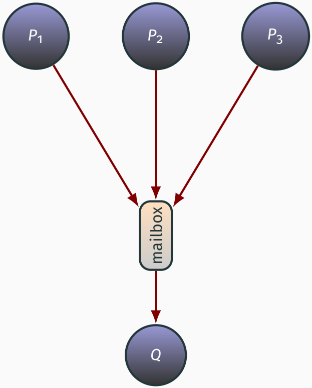
# Mailbox Types for Unordered Interactions

Ugo de'Liguoro Luca Padovani

Dipartimento di Informatica, Università di Torino, Italy

# Introduction

# 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**

# Example: Bank Transactions in Scala

```
class Account(var balance: Double) extends ScalaActor[AnyRef] {
  override def process(msg: AnyRef) {
    msg match {
      case dm: DebitMessage =>
        balance += dm.amount
        sender.send(new ReplyMessage())
      case cm: CreditMessage =>
        balance -= cm.amount
        recipient.send(new DebitMessage(self, cm.amount))
        receive {
          case rm: ReplyMessage =>
            sender.send(new ReplyMessage())
        }
      case _: StopMessage => exit()
      case message =>
        val ex = new IllegalArgumentException("Unsupported␣message")
        ex.printStackTrace(System.err)
} } }
```

# Pitfalls

## Protocol Violations

- ReplyMessage should be sent only during a transaction

## Unprocessed Messages

- StopMessage should be sent only if no more DebitMessage and CreditMessage are guaranteed to arrive

## Deadlocks

- a mediator (the bank) is necessary to successfully perform transactions between two accounts

# Key Ideas

1. Types describe **mailboxes** (not processes)
2. Subtyping embodies the **unordered** nature of mailboxes
3. Well-typed processes **break even**

# Mailbox Calculus

# Syntax of the Mailbox Calculus

**Asynchronous $\pi$-calculus + tagged messages + $\texttt{fail}/\texttt{free}$**

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

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

# Example: Locks

$$
\begin{aligned}
\mathsf{Idle}(lock) \triangleq\ & \texttt{free}\ lock.\texttt{done} \\
& + lock?\texttt{acquire}(user).(user!\texttt{reply}[lock] \mid \mathsf{Busy}[lock]) \\
& + lock?\texttt{release}.\texttt{fail}\ lock
\end{aligned}
$$

$$
\mathsf{Busy}(lock) \triangleq lock?\texttt{release}.\mathsf{Idle}[lock]
$$

- a lock is either idle or busy
- an idle lock **can** be acquired, but **cannot** be released
- a busy lock **must** be released

# Properties

**Definition**
*P* is *mailbox conformant* if $P \not\rightarrow^* \mathcal{C}[\texttt{fail}\ a]$

**Example (non-conformant process)**

$$\text{Idle}(lock) \mid lock\,!\,\texttt{release}$$

**Definition**
*P* is *deadlock free* if $P \rightarrow^* Q \not\rightarrow$ implies $Q \equiv \texttt{done}$

**Example (conformant but deadlocking process)**

$$\begin{array}{l} \text{Idle}(lock) \mid lock\,!\,\texttt{acquire}[user] \mid lock\,!\,\texttt{acquire}[user] \\ \mid user?\texttt{reply}(l_1).user?\texttt{reply}(l_2).(l_1\,!\,\texttt{release} \mid l_2\,!\,\texttt{release}) \end{array}$$

# Mailbox Types

# 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

## Examples

$$\mathsf{Idle}(\mathit{lock}) \triangleq \mathtt{free}\ \mathit{lock}.\mathtt{done}$$
$$+\ \mathit{lock}?\mathtt{acquire}(\mathit{user}).(\mathit{user}!\mathtt{reply}[\mathit{lock}] \mid \mathsf{Busy}[\mathit{lock}])$$
$$+\ \mathit{lock}?\mathtt{release}.\mathtt{fail}\ \mathit{lock}$$

$$\mathsf{Busy}(\mathit{lock}) \triangleq \mathit{lock}?\mathtt{release}.\mathsf{Idle}[\mathit{lock}]$$

**Example (mailbox of an idle lock)**

$$?\mathtt{acquire}[!\mathtt{reply}[!\mathtt{release}]]^*$$

**Example (mailbox of a busy lock)**

$$?(\mathtt{release} \cdot \mathtt{acquire}[!\mathtt{reply}[!\mathtt{release}]]^*)$$

## Typing Judgments

$$\Gamma \vdash P$$

### Intuition

- $\Gamma =$ messages **produced** by $P -$ messages **consumed** by $P$

### Consequences

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

# Typing Rules for Input/Output

$$u : !\mathtt{m} \vdash u!\mathtt{m}$$

$$\frac{\Gamma, u : ?E \vdash P}{\Gamma, u : ?(\mathtt{m} \cdot E) \vdash u?\mathtt{m}.P}$$

message arguments omitted for simplicity

# Typing Rules for Guards

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

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

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

# Tricky Cases for External Choices

$$?(A \cdot B + B \cdot C)$$ ☹

$$?(A \cdot B + C \cdot B)$$ ☺

$$?(B \cdot (A + C))$$ ☺

# Parallel Composition

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

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

parallel inputs are forbidden

# Subtyping

$$\frac{\Gamma, u : \sigma \vdash P}{\Gamma, u : \tau \vdash P} \quad \tau \leqslant \sigma$$

**Example (output contravariance)**

$$\frac{\Gamma, u : \,!E \vdash P}{\Gamma, u : \,!(E + F) \vdash P}$$

**Example (order irrelevance)**

$$\frac{\Gamma, u : \dagger(E \cdot F) \vdash P}{\Gamma, u : \dagger(F \cdot E) \vdash P}$$

# Example: Typing a Lock

$$\begin{aligned}\mathsf{Idle}(lock) &\triangleq \mathtt{free}\ lock.\mathtt{done}\\ &+ lock?\mathtt{acquire}(user).(user!\mathtt{reply}[lock] \mid \mathsf{Busy}[lock])\\ &+ lock?\mathtt{release}.\mathtt{fail}\ lock\end{aligned}$$

$$\mathsf{Busy}(lock) \triangleq lock?\mathtt{release}.\mathsf{Idle}[lock]$$

**where**

- idle $lock : ?\mathtt{acquire}[\cdots]^*$
- busy $lock : ?(\mathtt{release} \cdot \mathtt{acquire}[\cdots]^*)$

# Example: Typing a Lock

$$
\begin{aligned}
\mathsf{Idle}(\mathit{lock}) &\triangleq \mathtt{free}\ \mathit{lock}.\mathtt{done} \\
&+ \mathit{lock}?\mathtt{acquire}(\mathit{user}).(\mathit{user}!\mathtt{reply}[\mathit{lock}] \mid \mathsf{Busy}[\mathit{lock}]) \\
&+ \mathit{lock}?\mathtt{release}.\mathtt{fail}\ \mathit{lock}
\end{aligned}
$$

$$
\mathsf{Busy}(\mathit{lock}) \triangleq \mathit{lock}?\mathtt{release}.\mathsf{Idle}[\mathit{lock}]
$$

**where**

- idle $\mathit{lock} : ?\mathtt{acquire}[\cdots]^*$
  $= ?(\mathbb{1} + \mathtt{acquire}[\cdots] \cdot \mathtt{acquire}[\cdots]^* + \mathtt{release} \cdot \mathbb{0})$
- busy $\mathit{lock} : ?(\mathtt{release} \cdot \mathtt{acquire}[\cdots]^*)$

# Example: Typing a Lock



$$
\begin{aligned}
\mathsf{Idle}(lock) \triangleq\ & \texttt{free}\ lock.\texttt{done} \\
+\ & lock?\texttt{acquire}(user).(user!\texttt{reply}[lock] \mid \mathsf{Busy}[lock]) \\
+\ & lock?\texttt{release}.\texttt{fail}\ lock
\end{aligned}
$$

$$
\mathsf{Busy}(lock) \triangleq lock?\texttt{release}.\mathsf{Idle}[lock]
$$

**where**

- idle $lock : ?\texttt{acquire}[\cdots]^*$
  $= ?(\mathbb{1} + \texttt{acquire}[\cdots] \cdot \texttt{acquire}[\cdots]^* + \texttt{release} \cdot \mathbb{0})$
- busy $lock : ?(\texttt{release} \cdot \texttt{acquire}[\cdots]^*)$

# Example: Typing a Lock

$?(\texttt{acquire}[\cdots] \cdot \texttt{acquire}[\cdots]^*)$

$$
\begin{aligned}
\mathsf{Idle}(\mathit{lock}) &\triangleq \texttt{free}\ \mathit{lock}.\texttt{done} \\
&+ \mathit{lock}?\texttt{acquire}(\mathit{user}).(\mathit{user}!\texttt{reply}[\mathit{lock}] \mid \mathsf{Busy}[\mathit{lock}]) \\
&+ \mathit{lock}?\texttt{release}.\texttt{fail}\ \mathit{lock}
\end{aligned}
$$

$$\mathsf{Busy}(\mathit{lock}) \triangleq \mathit{lock}?\texttt{release}.\mathsf{Idle}[\mathit{lock}]$$

**where**

- idle $\mathit{lock} : ?\texttt{acquire}[\cdots]^*$
  $= ?(\mathbb{1} + \texttt{acquire}[\cdots] \cdot \texttt{acquire}[\cdots]^* + \texttt{release} \cdot \mathbb{0})$
- busy $\mathit{lock} : ?(\texttt{release} \cdot \texttt{acquire}[\cdots]^*)$

# Example: Typing a Lock

$$\mathsf{Idle}(lo \quad ?(\texttt{release}\cdot\mathbb{0}) \quad .\texttt{done} \qquad ?\mathbb{0}$$

$$+\ lo\ \ ?\texttt{acquire}(user).(\ \ ser!\texttt{reply}[lock] \mid \mathsf{Busy}[lock])$$

$$+\ lock?\texttt{release}.\texttt{fail}\ lock$$

$$\mathsf{Busy}(lock) \triangleq lock?\texttt{release}.\mathsf{Idle}[lock]$$

**where**

- idle $lock$ : $?\texttt{acquire}[\cdots]^*$
  $= ?(\mathbb{1} + \texttt{acquire}[\cdots]\cdot\texttt{acquire}[\cdots]^* + \texttt{release}\cdot\mathbb{0})$
- busy $lock$ : $?(\texttt{release}\cdot\texttt{acquire}[\cdots]^*)$

# Properties of Well-Typed Processes

**Theorem (conformance)**
*If $\Gamma \vdash P$, then $P$ is mailbox conformant*

**Lemma (type preservation)**
*If $\Gamma \vdash P$ and $P \rightarrow Q$, then $\Gamma \vdash Q$*

# Dependency Graphs

# Mailbox Dependencies

$$(\nu a)(\nu b)(a?\mathtt{m}.\mathtt{free}\ a.b!\mathtt{m} \mid b?\mathtt{m}.\mathtt{free}\ b.a!\mathtt{m}) \not\to$$

**Remark**

- this process is **mailbox conformant** but also **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*

# Typing Judgments, Refined

## Dependency Graphs

$$\varphi ::= \emptyset \mid \{u, v\} \mid \varphi \sqcap \varphi \mid (\nu a)\varphi$$

## Typing Judgments with Dependencies

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

## Example (refined rule for inputs)

$$\frac{\Gamma, u : ?E \vdash P :: \varphi}{\Gamma, u : ?(\mathtt{m} \cdot E) \vdash u?\mathtt{m}\,.\,P :: \bigsqcap_{v \in \mathrm{dom}(\Gamma)} \{u, v\}}$$

# Properties of Well-Typed Processes

**Theorem (deadlock freedom)**
*If* $\Gamma \vdash P :: \varphi$, *then* $P$ *is deadlock free*

**Definition (finitely unfolding process)**
$P$ is **finitely unfolding** if every maximal reduction of $P$ invokes recursive processes finitely many times

**Theorem (fair termination)**
*If* $\Gamma \vdash P :: \varphi$ *for* $P$ *finitely unfolding, then* $P \rightarrow^* Q$ *implies* $Q \rightarrow^* \mathsf{done}$

**Corollary (no garbage)**
*In a finitely unfolding process every message* ***can*** *be consumed*

# Concluding Remarks

# Summary

## Mailbox Calculus

- processes that communicate through **first-class mailboxes**
- subsumes the actor model

## Mailbox Types

- simple and intuitive semantics and typing rules
- mailbox conformance + mailbox bounds

## In the paper (ECOOP'18, draft on my home page)

- formal definitions and proofs
- more examples, encoding of binary sessions

# Further Developments

Application to **real-world languages**

- Java + annotations

Relation with **linear logic**?

- similarities between mailbox types and LL formulas
- most (but not all...) typing rules taken directly from LL

# Relating Mailbox Types to Linear Logic

## Interpretation of types

| $E$ | $\widehat{!E}$ | $\widehat{?E}$ |
|---|---|---|
| $\mathbb{0}$ | $0$ | $\top$ |
| $\mathbb{1}$ | $1$ | $\bot$ |
| $\mathtt{m}$ | $\mathtt{m}$ | $\mathtt{m}^\bot$ |
| $E + F$ | $\widehat{!E} \oplus \widehat{!F}$ | $\widehat{?E} \mathbin{\&} \widehat{?F}$ |
| $E \cdot F$ | $\widehat{!E} \otimes \widehat{!F}$ | $\widehat{?E} \mathbin{\invamp} \widehat{?F}$ |

## Simple facts

- $?E$ and $!E$ have dual interpretations
- $\sigma \leq \tau$ implies $\vdash \widehat{\tau}^\bot, \widehat{\sigma}$ derivable in (one sided) LL

# Relating Mailbox Types to Linear Logic

| judgement | behavior | choice | LL |
|---|---|---|---|
| $u : ?(A \cdot B) \vdash P$ | $P$ receives both A and B | internal | $\parr$ |
| $u : !(A \cdot B) \vdash P$ | $P$ sends both A and B | external | $\otimes$ |
| $u : ?(A + B) \vdash P$ | $P$ receives either A or B | external | $\&$ |
| $u : !(A + B) \vdash P$ | $P$ sends either A or B | internal | $\oplus$ |

# Relating Mailbox Types to Linear Logic

| judgement | behavior | LL |
|---|---|---|
| $u : ?\mathbb{1} \vdash P$ | $P$ deallocates $u$ | $\bot$ |
| $u : !\mathbb{1} \vdash P$ | $P$ discards $u$ | $1$ |
| $u : ?\mathbb{0} \vdash P$ | $P$ fails | $\top$ |
| $u : !\mathbb{0} \vdash P$ | — | $0$ |