Open Distributed Systems

- Addition of new components.
- Replacement of existing components.
- Changes in interconnections.

Actor Configurations

Model open system components:
- set of individually named actors.
- messages "on-route".
- interface to environment:
  - receptionists
  - external actors
Synchronous vs Asynchronous Communication

- Ti-Calculus (and other process algebras such as CCS, CSP) take synchronous communication as a primitive.
- Actors assume asynchronous communication is more primitive.
Communication Medium

- In PI-Calculus, channels are explicitly modelled. Multiple processes can share a channel, potentially causing interference.

- In the actor model, the communication medium is not explicit. Actors (active objects) are first-class, history-sensitive entities with an explicit identity used for communication.
Fairness

The actor model theory assumes fair computations:

1. message delivery is guaranteed.
2. individual actor computations are guaranteed to progress.

Fairness is very useful for reasoning about equivalences of actor programs but can be hard/expensive to guarantee; in particular when distribution and failures are considered.
PROGRAMMING LANGUAGES
INFLUENCED BY $\Pi$-CALCULUS
AND ACTORS.

- Scheme '75
- Act1 '87
- Acore '87
- Rosette '89
- Oblig '94
- Erlang '93
- ABCL '90
- SALSA '99
- Amber '86
- Facile '89
- CML '91
- Pict '94
- Nonadic Pict '99
- JOCAVL '99
Actor (Agent) Model

Actor

Thread

Message

Internal variables

(1)

Methods

State

(2)

Mailbox

(3)
AGHA, MAISON, SMITH & TALCOTT

1. Extend a functional language (λ-Calculus with tips + pairs) with actor primitives.

2. Define an operational semantics for actor configurations.

3. Study various notions of equivalence of actor expressions and configurations.

4. Assume fairness:
   - guaranteed message delivery.
   - individual actor progress.
\[ \text{\(\lambda\)-CALCULUS} \]

**Syntax**

\[ e ::= \lambda \quad v \quad \text{\(\lambda\)} v . e \quad (e \; e) \]

**Example**

\[ (\lambda x . x) \; 5 \]

\[ 5 \quad x \left\{ 5 / x \right\} \quad [5 / x] \; x \]
\( \text{pr} (x, y) \) returns a pair containing \( x \) & \( y \).

\( \text{ispr} (x) \) returns \( 1 \) if \( x \) is a pair; \( 0 \) otherwise.

\( \text{1st} (\text{pr} (x, y)) = x \)  
1st returns The first value \( x \).

\( \text{2nd} (\text{pr} (x, y)) = y \)  
2nd returns The second value.
A FOR PRIMITIVES

send(a, v)

sends value v to actor a.

new(b)

creates a new actor with behavior b, and returns the identity/name of the newly created actor.

ready(b)

becomes ready to receive a new message with behavior b.
**Actor Language Example**

\[ b_5 = \text{rec}(\forall y. \exists x. \text{seg}(\text{send}(x, 5), \text{ready}(y))) \]

receives an actor name \( x \) and sends the number 5 to that actor, then it becomes ready to process new messages with the same behavior \( y \).

**Sample Usage**

\[ \text{send(new(b_5), a)} \]

A SINK

\[ \text{sink = rec}(\forall b. \exists m. \text{ready}(b)) \]

an actor that disregards all messages.
cell = rec (\(\lambda b. \lambda c. \lambda m.\)
  if (get?(m),
    seg (send (cust (m), c),
      ready (b(c))),
    if (set?(m),
      ready (b(contents (m))),
      ready (b(c)))),

Using the cell:

let a = new(cell(0)) in
seg (send (a, mkset(3)),
  send (a, mkset(2)),
  send (a, mkget(c)))
Exercises

0. Write `get?`, `cust`, `set?`, `contents`, `mkset`, `mkget` to complete the reference cell example in the AMST actor language.

1. Modify `Bcell` to notify a customer when the cell value is updated (such as in the IT-calculus cell example).
Dining Philosophers in Actor Language

\[ \text{phil} = \text{rec } \lambda b. \lambda l. \lambda r. \lambda \text{self. } \lambda \text{sticks. } \lambda \text{m.} \]

\[
\text{if } (\text{eg? } (\text{sticks, 0}),
\hspace{1cm} \text{ready } (b(l, r, \text{self}, 1)),
\hspace{1cm} \text{seg } ( \text{send } (l, \text{mkrelease } (\text{self})),
\hspace{1cm} \text{send } (r, \text{mkrelease } (\text{self})),
\hspace{1cm} \text{send } (l, \text{mkpickup } (\text{self})),
\hspace{1cm} \text{send } (r, \text{mkpickup } (\text{self})),
\hspace{1cm} \text{ready } (b(l, r, \text{self}, 0))))
\]
chopstick = rec (\b. \h. \w. \m.
    if (pickup?(m),
        if (eq? (h, nil),
            seg (send (getphil (m), nil),
                ready (b (getphil (m), nil))),
            ready (b (h, getphil (m))),
        if (release?(m),
            if (eq? (w, nil),
                ready (b (nil, nil)),
                seg (send (w, nil),
                    ready (b (w, nil))),
            ready (b (h, w))))))
letrec c1 = new (chopstick (nil, nil)),
c2 = new (chopstick (nil, nil)),
p1 = new (phil (c1, c2, p1, 0)),
p2 = new (phil (c2, c1, p2, 0)) in e

where e is defined as:

e = seq (send (c1, mkpickup (p1)),
   send (c2, mkpickup (p1)),
   send (c1, mkpickup (p2)),
   send (c2, mkpickup (p2)))
Damn Philosophers in Actor LANG (4)

Auxiliary definitions:

\[ \text{mkpickup} = \lambda p. p \]
\[ \text{morelease} = \text{nil} \]
\[ \text{pickup?} = \lambda m. \text{not} (\text{eg?}(m, \text{nil})) \]
\[ \text{release?} = \lambda m. \text{eg?}(m, \text{nil}) \]
\[ \text{getphid} = \lambda m. m \]
Actor Garbage Collection

- Node 10 is blocked (idle).
- Node 12 is unblocked (non-idle).
- Node 4 is the root.