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 "en-route".
- interface to environment:
  * receptionists
  * external actors

SYNCHRONOUS vs ASYNCHRONOUS COMMUNICATION

- π-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 π-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 π-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
- Nomadic Pict '99
- JOCAML '99

AGHA, MASON, SMITH & TALCOTT

1. Extend a functional language $\left(\begin{array}{l}\lambda\text{-Calculus}\\ + \text{ifs} + \text{pairs}\end{array}\right)$ 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.

λ-CALCULUS

SYNTAX

$$
\begin{aligned}
e ::= \;& v \\
|\;& \lambda v . e \\
|\;& (e\ e)
\end{aligned}
$$

value

function abstraction

application

EXAMPLE

$(\lambda x . \underline{x}\ 5)$

5

$x\{5/x\}$ $\leftarrow \pi$

$[5/x]\,x$ $\leftarrow \lambda$

PAIRING PRIMITIVES

$pr(x,y)$ returns a pair containing x & y.

$ispr(x)$ returns t if x is a pair; f otherwise

$1st(pr(x,y)) = x$ 1st returns the first value of a pair

$2nd(pr(x,y)) = y$ 2nd returns the second value.

ACTOR PRIMITIVES

send(a,v) sends value v to actor a.

letactor{x:=b}e creates a new actor with behavior b, and binds variable x in expression e to the address of the newly created actor.

become(b) creates an anonymous actor to carry out the rest of the computation, and changes behavior to b.

ACTOR LANGUAGE EXAMPLES

$b5 = rec(\lambda y. \lambda x. seq(send(x, 5), become(y)))$

receives an actor name x and sends the number 5 to that actor, then it becomes an actor with the same behavior y.

SAMPLE USAGE

$e = letactor\{z := b5\}\ send(z, a)$

A SINK

$sink = rec(\lambda b. \lambda m. become(b))$

an actor that disregards all messages.

REFERENCE CELL IN ACTOR LANGUAGE

$$
\begin{aligned}
B_{cell} = \mathrm{rec}(\lambda b.\lambda c.\lambda m.\ & \mathrm{if}(\underline{\mathrm{get?}}(m), \\
& \quad \mathrm{seq}(\mathrm{become}(b(c)), \\
& \qquad \mathrm{send}(\underline{\mathrm{cust}}(m), c)) \\
& \quad \mathrm{if}(\underline{\mathrm{set?}}(m), \\
& \qquad \mathrm{become}(b(\underline{\mathrm{contents}}(m))) \\
& \qquad \mathrm{become}(b(c)))))
\end{aligned}
$$

Using the cell:

letactor $\{a := B_{cell}(0)\}\ e$

$$
\begin{aligned}
e = \mathrm{seq}(&\mathrm{send}(a, \underline{\mathrm{mkset}}(3)), \\
&\mathrm{send}(a, \underline{\mathrm{mkset}}(5)), \\
&\mathrm{send}(a, \underline{\mathrm{mkget}}(e)))
\end{aligned}
$$

EXERCISES

① Write get?
cust
set?
contents
mkset
mkget
to complete the reference cell example in the AMST actor language.

② Modify Bcell to notify a customer when the cell value is updated (such as in the Π-calculus cell example)

DINING PHILOSOPHERS IN A FIRST ACTOR LANGUAGE

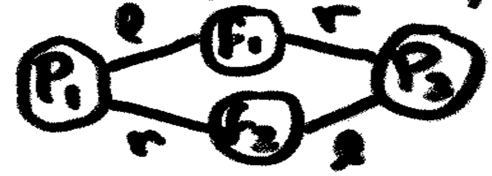
```
B_phil = rec(λb.λl.λr.λself.λacks.λm.
          if (eq?(m,self),
              if (eq?(acks,0),
                  become(b(l,r,self,acks+1)),
                  seq(send(l, mkrelease(self)),
                      send(r, mkrelease(self)),
                      become(b(l,r,self,0)),
                      send(l, mkpickup(self)),
                      send(r, mkpickup(self)))),
              become(b(l,r,self,acks))))
```

DINING PHILOSOPHERS IN AMST (2)

```
Bfork = rec (λb. λh. λw. λm.
            if (pickup?(m),
                if (eq?(h, nil),
                    seq(send(phil(m), phil(m)),
                        become(b(phil(m), nil))),
                    become(b(h, phil(m)))),
                if (release?(m),
                    if (eq?(w, nil),
                        become(b(nil, nil)),
                        seq(send(w, w),
                            become(b(w, nil)))),
                    become(b(h, w)))))
```

DINING PHILOSOPHERS IN AMST (3)

Using the definitions to set up a 2-phils dining table:



$$
\begin{aligned}
\text{letactor}\ \{\ & f_1 := B_{fork}(nil, nil), \\
& f_2 := B_{fork}(nil, nil), \\
& p_1 := B_{phil}(f_1, f_2, p_1, 0), \\
& p_2 := B_{phil}(f_2, f_1, p_2, 0)\ \}\ e
\end{aligned}
$$

where e is defined as:

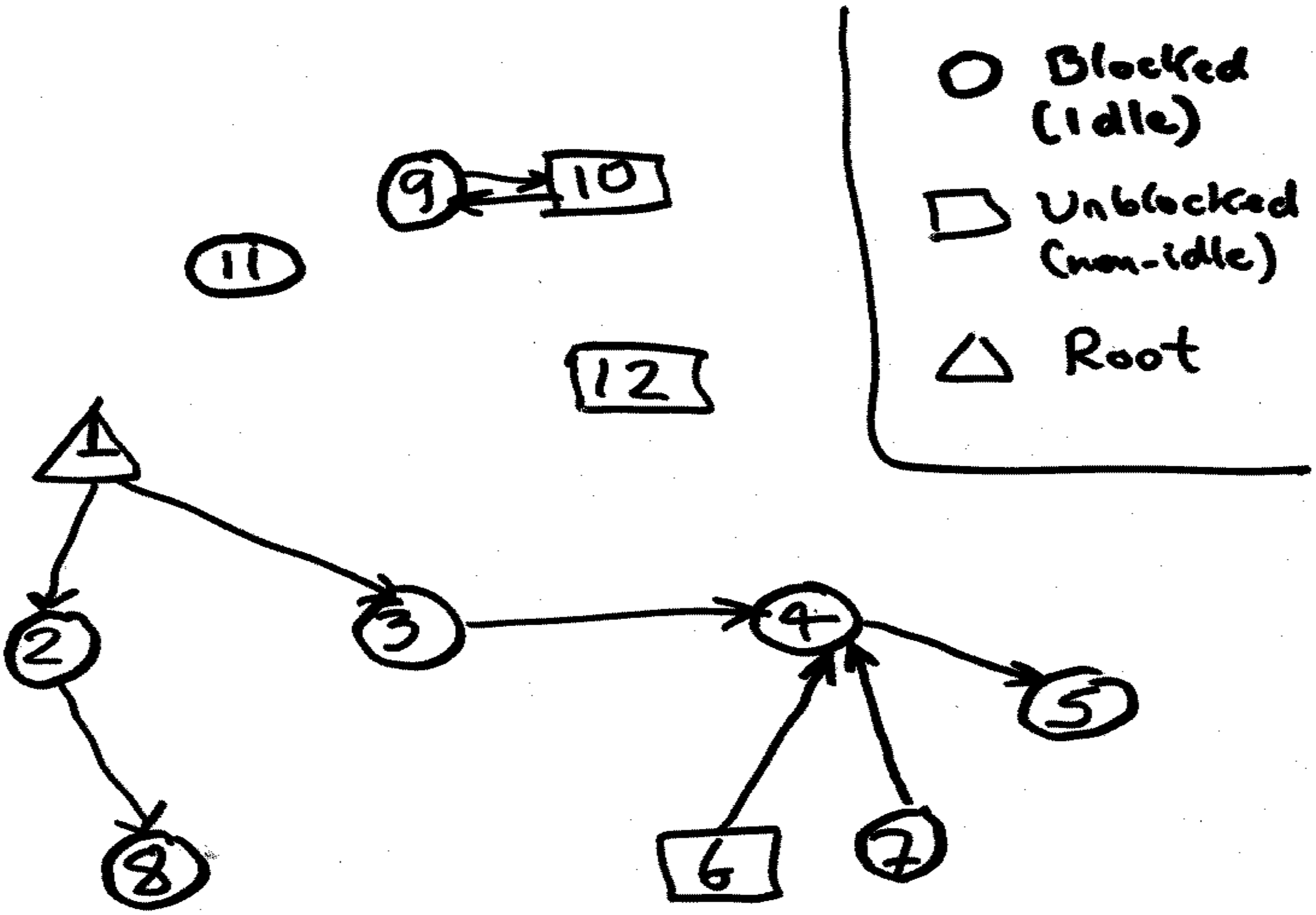
$$
\begin{aligned}
e = \ seq(\ & send(f_1, mkpickup(p_1)), \\
& send(f_2, mkpickup(p_1)), \\
& send(f_1, mkpickup(p_2)), \\
& send(f_2, mkpickup(p_2)))
\end{aligned}
$$

DINING PHILOSOPHERS IN AMST (4)

Auxiliary definitions:

```
mkpickup = λp. pr("pickup", p)
mkrelease = λp. pr("release", p)
pickup? = λm. if (ispr?(m),
                  eq?(1st(m), "pickup"),
                  nil)
release? = λm. if (ispr?(m),
                   eq?(1st(m), "release"),
                   nil)
phil = λm. if(pickup?(m),
              2nd(m),
              nil)
```

ACTOR GARBAGE COLLECTION



6A