CS 476 – Programming Language Design

William Mansky
Concurrency from Scratch

• We’ve seen two ways of adding concurrency to a simple imperative language.

• What would it look like to design a language with concurrency in mind?

• Main questions:
  — What is the state of a concurrent program?
  — What actions can a process perform?
  — How do processes communicate?
Concurrency from Scratch: Actor Model

code1
code2
code3
Concurrency from Scratch: Actor Model

code1'  code2  code3

code1 → code1'
Concurrency from Scratch: Actor Model

```
send(msg);

code1

code2

code3
```
Concurrency from Scratch: Actor Model

A
send(B, msg);
code1

B
code2

C
code3
Concurrency from Scratch: Actor Model

A: code1  B: msg  C: code3
Concurrency from Scratch: Actor Model

A

code1

B

x = recv();
f(x)

C

code3

B: msg
Concurrent from Scratch: Actor Model

A

spawn(body);
code1

B
code2

C
code3
Concurrency from Scratch: Actor Model
Concurrency from Scratch: Actor Model

A

\[ x = \text{spawn(body)}; \]
\[ \text{code1(x)} \]

B

\[ \text{code2} \]

C

\[ \text{code3} \]
Concurrency from Scratch: Actor Model

code1(D)
code2
code3

body
Actor Model

• The state of a program is a collection of *actors*

• Each actor can:
  — do sequential computation
  — send messages to other actors, if it knows their names
  — create new actors

• No shared variables or memory, just messages

• Communication is *asynchronous*

• Messages can contain actor names

• Used in Erlang, Pony, and various message-passing libraries
Simple Actor Language: Syntax

\[ E ::= \text{(as in a functional language)} \mid \text{self} \]

\[ C ::= \ldots \mid \text{send}(E, E) \mid \text{let } \langle \text{id} \rangle = \text{recv}() \mid \text{let } \langle \text{id} \rangle = \text{spawn}(C) \]
let proc1 = let x = recv();
    match x with (id, _) -> send(id, "Hi!")

let p = spawn(proc1);
send(p, (self, "Hello!"));
let answer = recv();
Simple Actor Language: Types?

\[ E := \text{(as in a functional language)} \]
\[ C := \ldots \mid \text{send}(E, E) \mid \text{let } \langle \text{ident} \rangle = \text{recv}() \mid \text{let } \langle \text{ident} \rangle = \text{spawn}(C) \]

\[
\frac{\Gamma \vdash e_1 : \text{pid} \quad \Gamma \vdash e_2 : \text{?}}{\Gamma \vdash \text{send}(e_1, e_2) : \text{ok}} \quad \quad \frac{\Gamma \vdash x : \text{?}}{\text{let } x = \text{recv}() : \text{ok}}
\]

\[
\frac{\Gamma \vdash x : \text{pid}}{\text{let } x = \text{spawn}(c) : \text{ok}}
\]
Simple Actor Language: Semantics

- An execution state has a set of *actors* and a pool of in-flight *messages*.
- An actor has an id, code, and local state.
- Every step involves only one actor.

\[
\begin{align*}
(i_j, c_j, \rho_j, M) \rightarrow (i_j, c'_j, \rho'_j, M') \\
\left[ (i_1, c_1, \rho_1); \ldots; (i_j, c_j, \rho_j); \ldots; (i_n, c_n, \rho_n) \right], M) \rightarrow \\
\left[ (i_1, c_1, \rho_1); \ldots; (i_j, c'_j, \rho'_j); \ldots; (i_n, c_n, \rho_n) \right], M' 
\end{align*}
\]
**Simple Actor Language: Semantics**

\[
\begin{align*}
(e_1, \rho) \downarrow j & \quad (e_2, \rho) \downarrow v \\
\hline
(i, \text{send}(e_1, e_2), \rho, M) & \rightarrow (i, \text{skip}, \rho, \{(j, v)\} \cup M)
\end{align*}
\]

\[
\begin{align*}
(i, x = \text{recv}(), \rho, \{(i, v)\} \cup M) & \rightarrow (i, \text{skip}, \rho[x \mapsto v], M)
\end{align*}
\]

\[
i' \text{ fresh} \\
\hline
(i, x = \text{spawn}(c), \rho, M) & \rightarrow \\
((i, \text{skip}, \rho[x \mapsto i']); (i', c, \{\text{self = } i'\}), M)
\]
Simple Actor Language: Example

let proc1 = let x = recv();
   match x with (id, _) -> send(id, “Hi!”)

let p = spawn(proc1); send(p, (self, “Hello!”));
let answer = recv();
Simple Actor Language: Example

let proc1 = let x = recv(); (1, {self = 1})
    match x with (id, _) -> send(id, "Hi!")

let p = spawn(proc1);
send(p, (self, "Hello!")); (0, ρ₀[p = 1])
let answer = recv();
Simple Actor Language: Example

let proc1 = let x = recv(); → (1, \{self = 1\})
    match x with (id, _) -> send(id, “Hi!”)

let p = spawn(proc1);

send(p, (self, “Hello!”)); → (1, (0, Hello!))

let answer = recv(); ← (0, ρ₀[p = 1])
Simple Actor Language: Example

let proc1 = let x = recv();
    match x with (id, _) -> send(id, "Hi!") → (1, \{ x = (0, "Hello!"))

let p = spawn(proc1);
send(p, (self, "Hello!"));
let answer = recv(); ← (0, ρ₀[p = 1])
Simple Actor Language: Example

let proc1 = let x = recv();
    match x with (id, _) -> send(id, "Hi!")
        → (1, {x = (0, "Hello")})

let p = spawn(proc1);
send(p, (self, "Hello!"));
let answer = recv(); → (0, p = 1)
Simple Actor Language: Example

```plaintext
let proc1 = let x = recv();
    match x with (id, _) -> send(id, "Hi!")
    (1, {x = (0, "Hello!")})

let p = spawn(proc1);
send(p, (self, "Hello!"));
let answer = recv();
    (0, ρ₀[p = 1, answer = "Hi!"])```