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

A: send(B, msg);
   code1

B: code2

C: code3
Concurrency from Scratch: Actor Model

A

code1

B

code2;
...

C

code3

B: msg
Concurrency from Scratch: Actor Model

A

code1

B

x = recv();
...x...

C

code3

B: msg
Concurrency from Scratch: Actor Model

A

code1

B

...msg...

C

code3
Concurrency from Scratch: Actor Model

A

spawn(body);
code1

B

code2

C

code3
Concurrency from Scratch: Actor Model

A: code1
B: code2
C: code3
D: 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

• Used in Erlang, Pony, and various message-passing libraries (e.g. Scala/Akka)
Simple Actor Language: Syntax

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

Values include \emph{names} of actors (like thread IDs)

\[
\text{send(actor2, 5); let x = recv()}
\]
\[
\text{send(actor2, (5, self))}
\]
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();
    match x with (id, _) -> send(id, “Hi!”)

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

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

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

\[
\Gamma \vdash e_1 : \text{pid} \quad \Gamma \vdash e_2 : ?
\]

\[
\Gamma \vdash \text{send}(e_1, e_2) : \text{ok}
\]

\[
\Gamma(x) = ?
\]

\[
\text{let } x = \text{recv()} : \text{ok}
\]

\[
\Gamma \vdash x : \text{pid} \quad \Gamma \vdash c : \text{ok}
\]

\[
\text{let } x = \text{spawn}(c) : \text{ok}
\]
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

\[
(i_j, c_j, \rho_j, M) \rightarrow (i_j, c'_j, \rho'_j, M')
\]

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

\[
\begin{align*}
(e_1, \rho) \Downarrow j & \quad (e_2, \rho) \Downarrow v \\
\frac{(i, \text{send}(e_1, e_2), \rho, M) \rightarrow (i, \text{skip}, \rho, \{j: v\} \cup M)}{(i, x = \text{recv}(), \rho, \{i: v\} \cup M) \rightarrow (i, \text{skip}, \rho[x \mapsto v], M)} &
\end{align*}
\]

\[
\begin{align*}
i' \text{ fresh} & \\
\frac{(i, x = \text{spawn}(c), \rho, M) \rightarrow ((i, \text{skip}, \rho[x \mapsto i']); (i', c, \{\text{self} = i'\})), M)}{}
\end{align*}
\]
let proc1 = let x = recv();
    match x with (id, _) -> send(id, "Hi!")

let p = spawn(proc1);
(0, \rho_0)
send(p, (self, "Hello!"));
let answer = recv();
Simple Actor Language: Example

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

let p = spawn(proc1);
send(p, (self, "Hello!") (0, \rho_0[p = 1])
let answer = recv();
```
Simple Actor Language: Example

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

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

1: (0, Hello!)
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, \rho_0[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])
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, answer = "Hi!"\})