Immmigration course on formal methods

Emilio Tuosto @ GSSI

Academic year 2020/2021

So far...

- An idea of FMs

Leonardo da Vinci

> “ Ma prima farò alcuna esperienza avanti ch’io più oltre proceda, perché mia intenzione è allegare prima l’esperienzia e poi colla ragione dimostrare. ”

eM’s (bad) translation

> “ Before proceeding further, I will first get some experiment, because my intention is to first understand the experiment and then to explain it with the intellect. ”

- Concurrency vs Parallelism
- Shared-memory

Message-passing

Pink Floyd

“Is there anybody out there?”

A glimpse of Erlang

```
ping(N, Pong_PID) ->
  Pong_PID ! {ping, self()},
  receive
    pong ->
      io:format("Ping received pong~n", [])
  end,
  ping(N - 1, Pong_PID).

ping(0, Pong_PID) ->
  Pong_PID ! finished,
  io:format("ping finished~n", []);
```

```
pong() ->
  receive
    finished ->
      io:format("Pong finished~n", []);
    {ping, Ping_PID} ->
      io:format("Pong received ping~n", []),
      Ping_PID ! pong,
      pong()
  end.
```

```
start() ->
  Pong_PID = spawn(example, pong, []),
  spawn(example, ping, [3, Pong_PID]).
```

Semantics

- Message passing
- FIFO buffers [[mailboxes in Erlang's jargon]]
- Spawn of threads

Asynchrony by design

Erlang is an embodiment of the well-known actor model of Hewitt and Agha...dates back to '73!

A glimpse of Erlang

```
ping(N, Pong_PID) ->
  Pong_PID ! {ping, self()},
  receive
    pong ->
      io:format("Ping received pong~n", [])
  end,
  ping(N - 1, Pong_PID).

ping(0, Pong_PID) ->
  Pong_PID ! finished,
  io:format("ping finished~n", []);
```

```
pong() ->
  receive
    finished ->
      io:format("Pong finished~n", []);
    {ping, Ping_PID} ->
      io:format("Pong received ping~n", []),
      Ping_PID ! pong,
      pong()
  end.
```

```
start() ->
  Pong_PID = spawn(example, pong, []),
  spawn(example, ping, [3, Pong_PID]).
```

Semantics

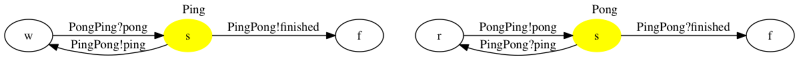
- Message passing
- FIFO buffers [[mailboxes in Erlang's jargon]]
- Spawn of threads

Asynchrony by design

Erlang is an embodiment of the well-known actor model of Hewitt and Agha...dates back to '73!

Friendlier representations

Local behaviour: communicating machines



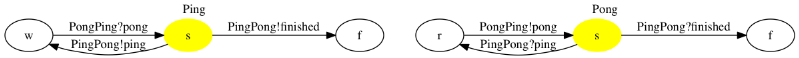
CFSMs (Brand & Zafiropulo 1983!): FIFO buffers as well

Choregraphy: global graph

...“synchronous” distributed workflow (Deniélou and Yoshida 2012)

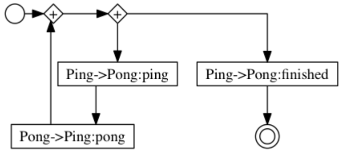
Friendlier representations

Local behaviour: communicating machines



CFSMs (Brand & Zafiropulo 1983!): FIFO buffers as well

Choregraphy: global graph



...“synchronous” distributed workflow (Deniélou and Yoshida 2012)

A glimpse of Erlang

```
ping(N, Pong_PID) ->
  Pong_PID ! {ping, self()},
  receive
    pong ->
      io:format("Ping received pong~n", [])
  end,
  ping(N - 1, Pong_PID).

ping(0, Pong_PID) ->
  Pong_PID ! finished,
  io:format("ping finished~n", []);
```

```
pong() ->
  receive
    finished ->
      io:format("Pong finished~n", []);
    {ping, Ping_PID} ->
      io:format("Pong received ping~n", []),
      Ping_PID ! pong,
      pong()
  end.
```

```
start() ->
  Pong_PID = spawn(example, pong, []),
  spawn(example, ping, [3, Pong_PID]),
  spawn(example, ping, [2, Pong_PID]).
```

Q:

Is this program correct?

A:

No!

Exercise:

find the bug

A glimpse of Erlang

```
ping(N, Pong_PID) ->
  Pong_PID ! {ping, self()},
  receive
    pong ->
      io:format("Ping received pong~n", [])
  end,
  ping(N - 1, Pong_PID).

ping(0, Pong_PID) ->
  Pong_PID ! finished,
  io:format("ping finished~n", []);
```

```
pong() ->
  receive
    finished ->
      io:format("Pong finished~n", []);
    {ping, Ping_PID} ->
      io:format("Pong received ping~n", []),
      Ping_PID ! pong,
      pong()
  end.
```

```
start() ->
  Pong_PID = spawn(example, pong, []),
  spawn(example, ping, [3, Pong_PID]),
  spawn(example, ping, [2, Pong_PID]).
```

Q:

Is this program correct?

A:

No!

Exercise:

find the bug

A glimpse of Erlang

```
ping(N, Pong_PID) ->
  Pong_PID ! {ping, self()},
  receive
    pong ->
      io:format("Ping received pong~n", [])
  end,
  ping(N - 1, Pong_PID).

ping(0, Pong_PID) ->
  Pong_PID ! finished,
  io:format("ping finished~n", []);
```

```
pong() ->
  receive
    finished ->
      io:format("Pong finished~n", []);
    {ping, Ping_PID} ->
      io:format("Pong received ping~n", []),
      Ping_PID ! pong,
      pong()
  end.
```

```
start() ->
  Pong_PID = spawn(example, pong, []),
  spawn(example, ping, [3, Pong_PID]),
  spawn(example, ping, [2, Pong_PID]).
```

Q:

Is this program correct?

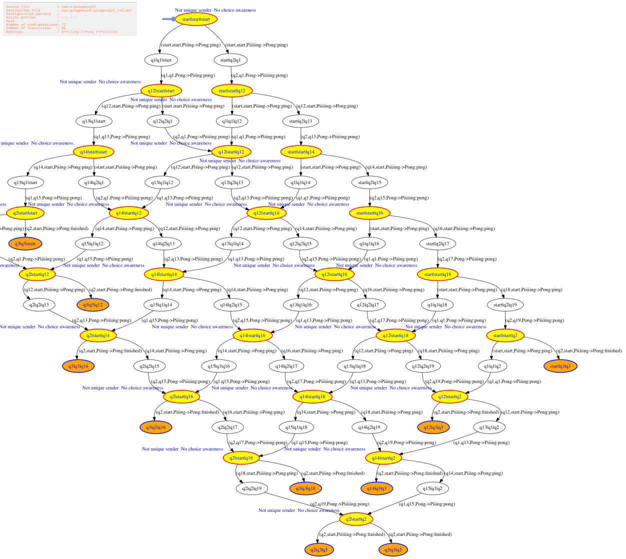
A:

No!

Exercise:

find the bug

Send ping-pong to shell !!! ... I mean, use ChoSyn



```
Source file              : tests/pingpong35
Destination file         : out/pingpong35/pingpong35_ts0.dot
Configuration pattern    :
Action pattern           : - - - -
Path                     :
Number of configurations: 72
Number of transitions    : 86
Machines                 : 0=Piiing 1=Pong 2=Piiiiing
```