Lecture 8: Functional programming and informal notation for processes

K. V. S. Prasad
Dept of Computer Science
Chalmers University

24 Sep 2011

Questions?

- Anything you want to say
  - Comments, questions, stray thoughts, etc.
  - Are we too fast/slow?
- Today’s ”and finally”
  - An introduction to academia

Plan for today

- Functional notation
  - Mostly for message passing
  - Examples, old and new
  - Informal notation, for illustration
    - Not expected to know this for exam
    - Translate to Ben-Ari style if you wish
    - Broadcast is not treated in Ben-Ari anyway

Chap 3 & 4 (skipped for now)

REMINDER: exercises in Chaps. 1, 2, 3, 6, 7, 8, 9

Factorial

```
fac 0 = 1
fac n = n * fac (n-1)  -- use if parm <> 0
```

In the context of this program of two defns.,

an expression is evaluated as follows: for a non-canonical term, find a matching pattern, and replace lhs by rhs

```
fac 3 = 3 * fac 2
      = 3 * (2 * fac 1)
      = 3 * (2 * (1 * fac 0))
      = 3 * (2 * (1 * 1))
      = 3 * (2 * 1)
      = 3 * 2
      = 6
```

Functional Programming (FP)

- (Pure) function has no side-effects
  - takes arguments and returns a value,
  - can use it wherever a value is expected
  - Not an imperative statement (command)
- In FP
  - A program is a set of definitions
  - To run this program, give it input: an expression
  - A run is the evaluation of this expression
- Anything you can do with Turing machines(TM), you can do with FP, and vice-versa

Processes and Functions

- I/O is imperative, so is every process
- Erlang is functional when computing values
  - but also has message send and receive.
  - They are imperative and have side-effects
  - Erlang mixes functional and imperative
    - even commands return values
    - so they can be used in expressions.
- Process definitions are like function definitions.
  - a process name can appear anywhere as a command
    - (an imperative statement).

CCS notation

- CCS = Calculus of Communicating Systems
  - Robin Milner et al (1980 on)
  - Very influential, worked on by hundreds at least
  - Uses synchronous channels
    - we address the channels, not the processes.
- Clock = tick! Clock
  - Output, pure channel
- Walk = left! right! Walk
  - Same machine in two states, or
  - L = left! R and R = right! L
    - machines that succeed each other

Values on channels

- Output takes a value
  - a(15)!, a(3*5)! and a(3*v)! all do the same if v=5
  - Can use constant, variable or expression
- Input takes a variable
  - a(x)? When I hear a value on channel a, I call it x
- Relay(a,b) = a(x)? b(x)! Relay (a, b)
  - Here the a, b are parameters to the process defn.

Cremona

- main = flash("Cremona”)! nap(3000). Main
  - The nap ’.’ represents passage of time
- Two messages at different intervals
  - p(m,t) = flash(m)! nap(t). p(m,t)
    main = p("beer", 5000) | p("cremona", 3000)

- The | means parallel composition
- Here, two incarnations of process p run in parallel.

Critical Section

- CS(d) = wait? print(d)! nap(100). signal! CS(d)
- Sem = wait! signal? Sem
- main = CS(l) | CS(r)

Counter

- Var (v) = incr? Var(v+1)
- Main = User(20) | User(20) | Var(0)
- User(0) = 0

  User(n) = incr! User(n-1)

- 0 appears first as an integer parameter, and then as a terminated process
- At the end, Var will have the value 40 because the process Var accepts no input while incrementing

Producer-consumer

```
P (n) =  ch(produce(n))! P(n+1)
C (xs) =  ch(x)? C(x:xs)
```

1. A list stores received values in the consumer.
2. Without the parameter to produce, it would yield the same element every time.

   functional programming => referential transparency

Multiplier process for matrix

M(v, n, s, e, w) =

    n(x)? e(y)? s(x)! w(y+v*x)! M(v, n, s, e, w)

1. The channels which say where the multiplier goes in the matrix are parameters.

   you can figure out the types.

2. The element value is v.
3. Draw state diagrams for the programs!

Multiplier process with selective input

```
M(v, n,s,e,w) = n(x)? N(x, v, n,s,e,w)
              + e(y)? E(y, v, n,s,e,w)
N(x, v, n,s,e,w) =  e(y)? s(x)! w(y+v*x)! M(v, n,s,e,w)
E(y, v, n,s,e,w) =   n(x)? s(x)! w(y+v*x)! M(v, n,s,e,w)
```

The + is a choice, guarded by which action happens.

If the north channel delivers first, M becomes N, if east, then E.

The repetition of code can be captured by another definition, but not by ”forking and rejoining” in functional style.

Draw state diagrams for the programs!

Dining philosophers

P(i) = think! f_i? f_i+1? eat! f_i! f_i+1! P(i)

F(i) = f_i! f_i? F(i)

1. The channels f_i pass no values (pure signals, empty envelopes).
2. I could write f(i), but that can be confused with a channel of name f passing a value i.

3. Introduce product notation for system?

ATM example

```
ATM =   enquire! answer(x)? print(x)! ATM
        + withdraw(c)! (ok? give(c)! ATM
                        +no? print(sorry)! ATM)

bank(d) = enquire? answer(d)! bank(d)
          + withdraw(c)? if c<=d then ok! bank(d-c)
                                 else no! bank(d)
```

1. ATM non-det. generates enquire or withdraw.
2. (Detail would show getting this command from the user).
3. Bank's choice because it can't predict user command.
4. Overlapping withdrawals not possible.

Barrier synchronisation with broadcast

N processes act in rounds, synchronise at start of each round.

p = start? done! P
barrier(0) = start! barrier(N)
barrier(n) = done? barrier (n-1)

1. System = barrier(0) | p | p |... p
   with N copies of p
2. start? p is defined as
   X = x? if x=start then p else X
3. A process that only wants to speak ignores all input

Barrier with sychronous channels

```
p = done! start? p

barrier(0) = starter(N)
barrier(n) = done? barrier (n-1)

starter(0) = barrier(N)
starter(n) = start! starter(n-1)

System = barrier(0) | p | p |... p
                    with N copies of p
```

Barrier with asynchronous channels

- Try the previous one
  - A fast process may jump the gun on the 2nd round by stealing the 1st ”start” of a slow starter
- So one way out is to number the procs and have separate start channels for each