Concurrent Programming intro (contd.)

K. V. S. Prasad
Dept of Computer Science
Chalmers University
21 January 2015

Plan for today

- Introduction to functional programming
- Example: Unit Record Equipment
- Radical concurrency
- State diagrams
- Concurrency models – (a)synchrony, time, …
- History
- Chaps from Ben-Ari (for 21, 23, and 26 Jan)
  - 1
  - 2.1 to 2.5
  - 3.1 to 3.5
  - 6.1 to 6.3

Introduction to functional programming

- Computation as reduction or simplification of expressions to canonical value
  - in the presence of definitions, reduction = replace left hand side by right hand side
- Different kinds of “variables”
  - Mathematics
    - Unknown – let x be the no. of apples
    - Placeholder – s = 16 t*t (give me a t value, I’ll give you the s)
  - Imperative programming
    - x:=1; while x < 10 do {x++; print x}
    - “how much is x?”.   When do you mean?

Factorial

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

In the context of this program of two definitions,

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
```

Example: Unit Record Equipment

- 1900's - 1950's – 1970's
  - Look up Wikipedia, etc.
- Typical application: payroll
  - One card per employee input (200 cpm)
  - Process info (100 records per min, avg)
  - Print salary info or cheque (300 lpm)
  - loop

```
read card;
process info;
print
```

But this is sequential. CDR waits while processing+printing

How to speed up?

Ex: URE 1

- We said the CDR waits. Do cards wait?
  - Active – passive distinction
    - Where does action come from?
      - Agents in nature. Why we see agents when there aren't any.
      - Animals vs plants+things
    - Are "objects" in CS active? No O-O in this course.
  - CDR, LPR and CPU act. Each has private memory.
    - How do they share info? How does the info move?
  - "Communication and Concurrency", Robin Milner.
    - Earlier version also from CTH library, but online.

Ex: URE 2

- CDR puts contents in shared memory
  - How does CPU know contents have arrived?
    - By interrupt, or by timing
      - Interrupt = check between instructions
  - What does CDR do meanwhile?
    - whole card is read and transferred as one?
    - If column by column, re-visit questions.

Ex: URE 3

- CDR
  loop
    c := card

  CPU
  loop
    p := f(c)

  LPR
  loop
    paper := p
- This is how we show parallel processes
  - But we need coordination/synchronisation/timing
  - CDR needs another c to read the next card into?
    - Is this an internal matter for CDR, and c is all we look at?
- CDR – c – CPU – p – LPR
  - So CPU can miss a card or re-read the same one.

Ex: URE 4

- ->
  CDR c CPU
  <-
- We try to work with signals (taps on shoulder)
  - Assume that reading a card and processing it take much longer than assigning to and from c, and sending and receiving signals

Ex: URE 5

-
CDR	CPU	LPR
loop	loop	loop
CR?	CF?	LF?
	LR?	
c := card	p := f(c)	paper := p
CF!	LF!	LR!
	CR!	

- Assuming signals are quick, and access to c and p are unguarded (why the post office sends you a small note to say a big parcel has arrived).
- All waiting to start with – deadlock. Kick start.

Concurrent? Parallel?

- Examples:
  - Max
    - Using handshake, broadcast
  - Sort
    - Using broadcast
  - Eight queens
- Crossing a door, sharing a printer

Some observations

1. Concurrency is simpler!
   a. Don't need explicit ordering
   b. The real world is not sequential
   c. Trying to make it so is unnatural and hard
      a. Try controlling a vehicle!
2. Concurrency is harder!
   1. many paths of computation (bank example)
   2. Cannot debug because non-deterministic
      so proofs needed
3. Time, concurrency, communication are issues

Terminology

- A ”process” is a sequential component that may interact or communicate with other processes.
- A (concurrent) ”program” is built out of component processes
- The components can potentially run in parallel, or may be interleaved on a single processor. Multiple processors may allow actual parallelism.

Interleaving

- Each process executes a sequence of atomic commands (usually called ”statements”, though I don’t like that term).
- Each process has its own control pointer, see 2.1 of Ben-Ari
- For 2.2, see what interleavings are impossible

Why arbitrary interleaving?

- Multitasking (2.8 is a picture of a context switch)
  - Context switches are quite expensive
  - Take place on time slice or I/O interrupt
  - Thousands of process instructions between switches
  - But where the cut falls depends on the run
- Runs of concurrent programs
  - Depend on exact timing of external events
  - Non-deterministic! Can't debug the usual way!
  - Does different things each time!

Arbitrary interleaving (contd.)

- Multiprocessors (see 2.9)
  - If no contention between CPU's
    - True parallelism (looks like arbitrary interleaving)
  - Contention resolved arbitrarily
    - Again, arbitrary interleaving is the safest assumption

But what is being interleaved?

- Unit of interleaving can be
  - Whole function calls?
  - High level statements?
  - Machine instructions?
- Larger units lead to easier proofs but make other processes wait unnecessarily
- We might want to change the units as we maintain the program
- Hence best to leave things unspecified

Course material

- Shared memory from 1965 – 1975 (semaphores, critical sections, monitors)
  - Ada got these right 1980 and 1995
  - And Java got these wrong in the 1990's!
- Message passing from 1978 – 1995
  - Erlang is from the 1990's
- Blackboard style (Linda) 1980's
- Good, stable stuff. What's new?
  - Machine-aided proofs since the 1980's
  - Have become easy-to-do since 2000 or so

Operating Systems (60's thru 70's)

- Divided into kernel and other services
  - which run as processes
- The kernel provides
  - Handles the actual hardware
  - Implements abstractions
    - Processes, with priorities and communication
  - Schedules the processes (using time-slicing or other interrupts)
- A 90's terminology footnote
  - When a single OS process structures itself as several processes, these are called "threads"

Goals of the course

- covers parallel programming too – but mostly about concurrency
- Classic problems of concurrent programming
  - mostly synchronisation problems
- Programming language constructs evolved for concurrent programming
- Practical knowledge of the programming techniques of modern concurrent programming languages

Application areas

- Introduction to the problems common to many computing disciplines:
  - Operating systems
  - Distributed systems
  - Real-time systems
  - Embedded systems
  - Networking