The Semantics of Concurrent Programming

K. V. S. Prasad
Dept of Computer Science
Chalmers University
February – March 2013

Semantics

- What do you want the system to do?
- How do you know it does it?
- How do you even say these things?
  - Various kinds of logic
- Build the right system (Validate the spec)
- Build it right (verify that system meets spec)

Mostly chapters 2, 3, 4 of

- M. Ben-Ari, ”Principles of Concurrent and Distributed Programming”, 2nd ed

  Addison-Wesley 2006

Recap and program

- You have used Java, JR and Erlang
  - As implementation languages in the labs
  - As vehicles for discussion in class
- What next?
  - For discussion, pseudo-code as in book
  - Reasoning using state diagrams and formal logic
  - But still no machine-aided proofs in course ☹
  - Examinable stuff will be from the textbook
    - I will present other motivating material, which will not be examined.

Concurrency: research history 1

- 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
  - CSP (1978), CCS (1980) : Wow, can just I/O do it all?
  - 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

From research to practice

- My dates refer to concurrency research
  - Strong sequential mindset in CS (as Java shows)
    - so take up lags by decades (C++ now)
    - Ignorance of concurrency work
    - assumption that it's easy (Therac)

Examples of CCS or CBS (make your own notes)

1. Natural examples (why not program like this?)
    1. Largest of multiset by handshake
    2. Largest of multiset by broadcast
    3. Sorting children by height
2. Occurring in nature (wow!)
    1. Repressilator
3. Actual programmed systems (boring)
    1. Shared bank account
        1. Don’t interleave between load and store

Radical Concurrency

- Don't start from sequential computation
- Handshake (kids meeting one-on-one)
  - Or like telephone, rendezvous
  - Can only happen when both parties present
    - Either waits for the other
    - With no data, symmetry between sender/receiver
- Broadcast
  - Speaker autonomous
  - Others must hear whatever spoken, whenever
- Our examples – concurrent, parallel, non-deterministic

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

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

State diagrams and scenarios

- Ben-Ari 5 -11, 16 -20, 22 – 24, 28 & 35-36
- In slides 2.4 and 2.5, note that the state describes variable values before the current command is executed.
- In 2.6, note that the ”statement” part is a pair, one statement for each of the processes
- Not all thinkable states are reachable from the start state

The standard Concurrency model

1. What world are we living in, or choose to?
   a. Synchronous or asynchronous?
   b. Real-time?
   c. Distributed?
2. We choose an abstraction that
   a. Mimics enough of the real world to be useful
   b. Has nice properties (can build useful and good programs)
   c. Can be implemented correctly, preferably easily

Obey the rules you make!

1. For almost all of this course, we assume single processor without real-time (so parallelism is only potential).
2. Real life example where it is dangerous to make time assumptions when the system is designed on explicit synchronisation – the train
3. And at least know the rules! (Therac).

To get started:

- What is computation?
  - States and transitions
  - Moore/Mealy/Turing machines
  - Discrete states, transitions depend on current state and input
- What is ”ordinary” computation?
  - Sequential. Why? Historical accident?

Example: the Frogs

- Slides 39 – 42 of Ben-Ari
- Pages 37 – 39 in book

Scenarios

- A scenario is a sequence of states
  - A path through the state diagram
  - See 2.7 for an example
  - Each row is a state
    - The statement to be executed is in bold

Transitions can be labelled

- (Discrete) computation = states + transitions
  - Both sequential and concurrent
    - Can two frogs move at the same time?
  - We use labelled or unlabelled transitions
    - According to what we are modelling
    - Chess games are recorded by transitions alone (moves)
      - States used occasionally for illustration or as checks

The Critical Section Problem

- Attempts to solve them
  - without special hardware instructions
    - Assuming load and store are atomic
  - Designing suitable hardware instructions
- Why study the problem without special instructions?
  - Case study of concurrency problems
  - Case study of proof methods

Requirements and Assumptions

- Correctness
  - Both p and q cannot be in their CS at once (mutex)
  - If p and q both wish to enter their CS, one must succeed eventually (no deadlock)
  - If p tries to enter its CS, it will succeed eventually (no starvation)
- Assumptions
  - A process in its CS will leave eventually (progress)
  - Progress in non-CS optional

Comments

- Pre- and post-protocols
  - These don’t share local or global vars with the rest of the program
- The CS models access to data shared between p and q

First try (alg 3.2, slide 3.3)

- The full state diagram shows only 16 states are reachable, out of 32
- These exclude states (p3,q3,*) so mutex is OK.
- The abbreviated program reduces state space
- if p1 is stuck in NCS with turn=1, q starves
- Deadlock free in the sense that p can enter CS
- Error: p and q both set and test ”turn”; if one dies the other is stuck