Chalmers | GÖTEBORGS UNIVERSITET
K. V. S. Prasad, Computer Science and Engineering

Concurrent Programming TDA383/DIT390

Saturday 22 Oct 2016 pm in HA/HB/HC
(example solutions)

Question 1. (a) q1, q2, q3, p1, q1, leading to n=0
(b) p1, q1, q2, p2, q3, p1, q1 giving n=1
(c) p1, p2, and then repeat (q1, q2, p1, p2, p1, p2) infinitely often.
(d) Yes (both processes run infinitely often).

Question 2. (a) See textbook, p. 152.
(b) See textbook, p. 119.
(c) Three, one for each condition variable, and one for monitor entry.
(d) Like (a), but remove condition variables, guard entry `put` by `when buffer not full` and entry `get` by `when buffer not empty`. The bodies of the entries need no `if` or `signal`.

Question 3. (a) The completed table:

	State = (pi, qi, wp, wq)	next state if p moves	next state if q moves
s_1	(p2, q2, 0, 0)	s_7 = (p3, q2, 1, 0)	s_2 =(p2, q3, 0, -1)
s_2	(p2, q3, 0, -1)	s_8 =(p3, q3, -1, -1)	s_4 =(p2, q5, 0, -1))
s_3	(p2, q3, 0, 1)	s_{11} =(p3, q3, 1, 1)	s_5 =(p2, q5, 0, 1)
s_4	(p2, q5, 0, -1)	s_{12} =(p3, q5, -1, -1)	s_1 =(p2, q2, 0, 0)
s_5	(p2, q5, 0, 1)	s_{13} =(p3, q5, 1, 1)	s_1 =(p2, q2, 0, 0)
s_6	(p3, q2, -1, 0)	s_{14} =(p5, q2, -1, 0)	s_9 =(p3, q3, -1, 1)
s_7	(p3, q2, 1, 0)	s_{15} =(p5, q2, 1, 0)	s_{10} =(p3, q3, 1, -1)
s_8	(p3, q3, -1, -1)	blocked	s_{12} =(p3, q5, -1, -1)
s_9	(p3, q3, -1, 1)	s_{16} =(p5, q3, -1, 1)	blocked
s_{10}	(p3, q3, 1, -1)	s_{17} =(p5, q3, 1, -1)	blocked
s_{11}	(p3, q3, 1, 1)	blocked	s_{13} =(p3, q5, 1, 1)
s_{12}	(p3, q5, -1, -1)	blocked	s_6 =(p3, q2, -1, 0)
s_{13}	(p3, q5, 1, 1)	blocked	s_7 =(p3, q2, 1, 0)
s_{14}	(p5, q2, -1, 0)	s_1 =(p2, q2, 0, 0)	s_{16} =(p5, q3, -1, 1)
s_{15}	(p5, q2, 1, 0)	s_1 =(p2, q2, 0, 0)	s_{17} =(p5, q3, 1, -1)
s_{16}	(p5, q3, -1, 1)	s_3 =(p2, q3, 0, 1)	blocked
s_{17}	(p5, q3, 1, -1)	s_2 =(p2, q3, 0, -1)	blocked

(b) There are no states with $(p5, q5, ?, ?)$.
(c) There are no states with both p and q `blocked`.
(d) Write S_i for the set of states that must lead to a $p5$-state in i moves. Then $S_0 = \{s_{14}$ through $s_{17}\}$, $S_1 = \{s_9, s_{10}\}$, $S_2 = \{s_6, s_7\}$, $S_3 = \{s_{12}, s_{13}\}$, $S_4 = \{s_8, s_{11}\}$. Then S is the union of these states. Each of the remaining states has a p-move into S, so a fair scheduler has to take one of them some time.

Question 4. (a) N holds at s_1 by initialisation. Subsequently, p can only reach $p2$ from $p5$. At $p3$ and $p5$, wp is -1 or 1 because of $p2$.
(b) M holds at s_1 since neither $p5$ nor $q5$ holds. Any state $(p5, q3, ?, ?)$ must have been reached from $(p3, q3, ?, ?)$ or $(p5, q2, ?, ?)$. In any $(p3, q3, ?, ?)$ state precisely one of p and q can proceed (there are four combinations of wp and wq, two of which have $wp = wq$ while the other two have $wp \neq wq$). Finally, if q moves in $(p5, q2, ?, ?)$, it will set wq to block itself.
(c) We show that any state $(p3, ?, ?, ?)$ must lead to a $p5$-state.

In the states $(p3, q2, ?, ?)$. either p moves so we have progress, or, after $q2$, we have $wp \neq wq$, so q blocks itself and releases p. Then the scheduler has to pick p.

In the states $(p3, q3, ?, ?)$, either q is blocked and p free, so we have progress, or the other way around. Then q must do $q5$ and release p, subsequently blocking itself anyway at $q2$.

Similarly for $(p3, q5, ?, ?)$.

Question 5. (a) The computation starts off `Ints --qi--> Sift`.

It prints 2 and evolves to `Ints --qi--> Filter(2) --q2--> Sift`, where `q2` is the local channel `q` when `Sift` received 2. We show only the integer parameter of Filter, the two channels are shown in the diagram.

`Filter(2)` then swallows 3 but relays 4, which `Sift` prints. New state: `Ints --qi--> Filter(2) --q2--> Filter(4) --q4--> Sift`

Filter(2) then swallows 3 but relays 4, which `Sift` prints. New state: `Ints --qi--> Filter(2) --q2--> Filter(4) --q4--> Sift`

Next, 5 and 7 are swallowed by `Filter(2)` and 6 by `Filter(4,)`, but 8 is printed and we have `Ints --qi--> Filter(2) --q2--> Filter(4) --q4--> Filter(8) --q8--> Sift`.

The program prints powers of 2.
(b) The program prints 3, 6, 12, 24 ... and 5, 10, 20, 40 ...
(c) and (d). No. The program grows forever, and greater channel capacity merely stores more numbers in the pipeline. No synchronisation nor do the actions change when a channel delivers a number.
(e) Change the line `if n mod p = 0` in `Filter` to `if n mod p ≠ 0`.

Question 6. (a) For barrier synchronisation. Each divisor is run past each remaining candidate prime before we go on to the next one.
(b) A `Worker` might finish with one divisor and steal the next `start` and re-do the same divisor, while another `Worker` misses that divisor altogether.
(c) When the `Master` has used up all the divisors, it times out and terminates, leaving the `Workers` waiting for the next `start`.
(d) `Master` posts a new tuple `todo` right at the start, and removes it at `alldone:`. Before it removes the `start`, each `Worker` tries to read `todo` with a time-out causing termination.