The Midterm Exam: Comments & Solutions

EPFL, LPD

STiDC'07

General issues

- Using disallowed objects (queues, etc.)
- No algorithm or no description
- Waiting

Problem 1

SRSW regular register $\rightarrow$ MRSW atomic register

(see the lecture slides for a solution)

Problem 2

Remove line 6 of *Read()* from Tromp's algorithm and show that the algorithm is incorrect using an execution with at most two invocations of *Read()*.
(see the updated lecture slides for a solution)

Problem 3

Binary consensus $+$ registers $\rightarrow$ multi-valued consensus

Main idea

Using bits (binary consensus) encode:

1. Process id $\Rightarrow$ find the "winner" among processes that participate, or
2. One of the proposed values.

Simple solution

Notation: N processes, D – (finite) domain of values

Assume: $D = \{1, \ldots, K\}$ (K finite)

We use: $R[1, \ldots, K]$ – registers, $C[1, \ldots, K]$ – binary consensus objects

upon *propose*(v) **do**

  $R[v] \leftarrow true$

  for $k \leftarrow 1$ **to** K **do**

    $b \leftarrow R[v]$

    if $C[k].propose(b)$ **then return** k

Problem 4

Same as Exercise 4: implement adaptive snapshot, i.e., atomic snapshot with step complexity $f(K)$
(K – the number of processes that use the snapshot)

Non-adaptive Snapshot

upon $scan_i$ **do**

- $t_1 \leftarrow collect()$, $t_2 \leftarrow t_1$
- **while** *true* **do**
  - $t_3 \leftarrow collect()$
  - **if** $t_3 = t_2$ **then return** $\langle t_3[1].val, \ldots, t_3[N].val \rangle$
  - **for** $k \leftarrow 1$ **to** N **do**
    - **if** $t_3[k].ts \geq t_1[k].ts + 2$ **then return** $t_3[k].snapshot$
  - $t_2 \leftarrow t_3$

procedure $collect()$

- **for** $k \leftarrow 1$ **to** N **do**
  - $x[k] \leftarrow R[k]$
- **return** x

Non-adaptive Snapshot (2)

procedure $update_i(v)$

$ts \leftarrow ts + 1$

$snapshot \leftarrow scan()$

$R[i] \leftarrow \langle ts, v, snapshot \rangle$

Adaptive Update

procedure *update*(v)

- **if** $myreg = \bot$ **then**
  - $myreg \leftarrow obtain()$
- $ts \leftarrow ts + 1$
- $snapshot \leftarrow scan()$
- $R[myreg] \leftarrow \langle ts, v, snapshot \rangle$

Adaptive Scan

```
upon scan_i do
    t_1 ← collect(), t_2 ← t_1
    while true do
        t_3 ← collect()
        if t_3 = t_2 then return ⟨ t_3[1].val, ..., t_3[t_3.length].val ⟩
        for k ← 1 to t_3.length do
            if t_3[k].ts ≥ t_1[k].ts + 2 then return t_3[k].snapshot
        t_2 ← t_3
```

A Disallowed Solution

procedure $obtain()$
$\quad myreg \leftarrow C.fetch\&inc()$

procedure $collect()$
$\quad$ **for** $k \leftarrow 1$ **to** $C.read()$ **do**
$\quad\quad x[k] \leftarrow R[k]$
$\quad$ **return** x

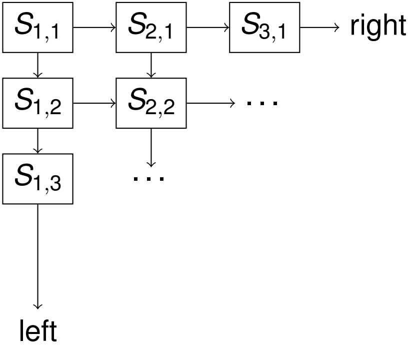
But we can use only registers!

The Splitter Object

- One operation: *splitter*
- Returns: *stop*, *left* or *right*
- If a single process executes *splitter*, then *stop* is returned.
- If two or more processes invoke *splitter*, then not all get the same output.
- At most one process gets *stop*.

Main Idea of Adaptive Snapshot

- Matrix of registers and splitters
- To obtain a register, a process must find a splitter that returns *stop*.
- Process starts from left top corner and follows the output of splitters.



The Obtain Operation

```
procedure obtain()
    x ← 1, y ← 1
    while true do
        s ← S[x, y].splitter()
        if s = "stop" then myreg ← ⟨ x, y ⟩
        else if s = "left" then y ← y + 1
        else x ← x + 1
```

The Collect Operation

procedure *collect*
- $C \leftarrow \langle\,\rangle$
- $d \leftarrow 1$
- **while** *diagonal d has a splitter that has been traversed* **do**
  - $C \leftarrow C \cdot \langle$ values of all non-$\perp$ registers on diagonal $d\ \rangle$
  - $d \leftarrow d + 1$
- **return** C