A Solution for the Exercise 8 (Faulty Base Objects)

EPFL, LPD

STiDC'08

Problem 1: Compare and Swap

Question: Is it possible to implement C&S using a finite number of base C&S objects one of which can be faulty in a non-responsive way?

Short answer: No, it is not.

Main Idea

- Problem P: implement C&S using base C&S objects, one of which can be non-responsive, and registers (non-faulty).
- Reduce to problem Q: implement consensus using registers in a system of $n > 1$ processes, one of which can crash $\Rightarrow$ impossible
- By contradiction: assume there exists an algorithm A that solves P using k C&S objects, in a system of n processes (one of which can crash)
- If we find an algorithm B that solves problem Q, using $A \Rightarrow$ contradiction

The Reduction

- We implement consensus in a system of $N = \max(k, n)$ processes, one of which can crash
- A process p_i that proposes a value, writes the value in a register $R[i]$ and waits until a decided value is written in register D:

initially: $D = \perp$, $R[1, \ldots, N] = \perp$

upon $propose_i(v)$ **do**
  $R[i] \leftarrow v$
  wait until $D \neq \perp$
  return D

The Reduction (2)

k processes emulate base C&S objects (in a parallel task; operation requests and responses passed via registers CS between each pair of processes):

parallel task C_i

- **initially**: $q = \perp$ (local variable)
- **while** *true* **do**
  - **for** $j \leftarrow 1$ **to** n **do**
    - $(type, oldval, newval) \leftarrow CS[i][j]$
    - **if** $type =$ `invocation` **then**
      - **if** $q = oldval$ **then** $q \leftarrow newval$
      - $CS[i][j] \leftarrow (\texttt{response}, q)$

The Reduction (3)

n processes run the following algorithm in a parallel task:

1. Wait until some value $v \neq \perp$ is written in some register $R[j]$,
2. Run algorithm A with operation $C\&S(\perp, v)$, using the emulated base C&S objects,
3. Write the value returned by A into register D.

Problem 2: SWMR Register

Problem: Implement SWMR register out of base SWMR registers, t of which can fail in a non-responsive way.

Solution: Use $2t + 1$ base registers, so that always majority is correct. Read/write from/to majority of registers.

The Idea

uses: $R[1, \ldots, 2t+1]$ – SWMR registers t of which can be non-responsive

upon $write_1(v)$ **do**
- $ts \leftarrow ts + 1$
- invoke $write_1(ts, v)$ on all $R[1, \ldots, 2t+1]$
- wait for $t + 1$ responses

upon $read_i$ **do**
- invoke $read_i(v)$ on all $R[1, \ldots, 2t+1]$
- wait for $t + 1$ responses
- **return** *the value v with the highest timestamp ts*

This algorithm implements a regular SWMR register!

The Idea (contd.)

The presented algorithm implements a regular SWMR register. However, a regular register can be transformed into an atomic one (see the lecture slides about register transformations).