CS451 (Distributed Algorithms): Lattice Agreement

1 SYSTEM MODEL & PRELIMINARIES

Processes. We assume a static system $\Psi = \{P_1, ..., P_n\}$ of $n = 2f + 1$ processes among which some processes can fail by *crashing*. A process that fails is said to be *faulty*; a non-faulty process is *correct*. We assume that at most f processes can fail. Processes communicate by exchanging messages over an authenticated point-to-point network. The communication network is *reliable*: if a correct process sends a message to a correct process, the message is eventually received.

Asynchrony. The processes are asynchronous: a process proceeds at its own arbitrary (and non-deterministic) speed. Moreover, the communication network is asynchronous: message delays are finite, but arbitrarily big. In other words, if a correct process sends a message to another correct process, it is not known when the message will be received; it is just known that the message will eventually be received.

Lattice agreement. Let $\mathcal{V}$ denote the set of values. The lattice agreement problem allows processes to agree on "similar" decisions despite failures. The lattice agreement problem exposes the following interface:

- **request** propose($I \subseteq \mathcal{V}$): a process proposes a set I.
- **indication** decide($O \subseteq \mathcal{V}$): a process decides a set O.

The proposal of a process P_i is denoted by I_i, whereas the decision of a process P_i is denoted by O_i.

The lattice agreement problem requires the following properties to be satisfied:

- *Validity:* Let a process P_i decide a set O_i. Then:
  - $I_i \subseteq O_i$, and
  - $O_i \subseteq \bigcup_{j \in [1,n]} I_j$.
- *Consistency:* Let a process P_i decide a set O_i and let a process P_j decide a set O_j. Then, $O_i \subseteq O_j$ or $O_i \supset O_j$.
- *Termination:* Every correct process eventually decides.

The validity property states that (1) the decided set must include the proposal set, and (2) the decided set includes the proposals of other processes (i.e., the decided set cannot include values which were not proposed). Consistency claims that decided values must be comparable. Finally, termination states that all correct processes eventually decide.

2 ALGORITHM

The algorithm considers two roles: proposers and acceptors. Every process plays *both* roles; the separation is included solely for the simplicity of the presentation.

Algorithm 1 Lattice Agreement Algorithm: Pseudocode of proposer P_i

upon init:
    Boolean $active_i = false$
    Integer $ack\_count_i = 0$
    Integer $nack\_count_i = 0$
    Integer $active\_proposal\_number_i = 0$
    Set $proposed\_value_i = \bot$

upon propose(Set *proposal*):
    $proposed\_value_i \leftarrow proposal$
    $active_i \leftarrow true$
    $active\_proposal\_number_i \leftarrow active\_proposal\_number_i + 1$
    $ack\_count_i \leftarrow 0$
    $nack\_count_i \leftarrow 0$
    trigger *beb*.broadcast$(\langle \textsc{proposal}, proposed_value_i, active_proposal_number_i \rangle)$

upon reception of $\langle \textsc{ack}$, Integer $proposal\_number \rangle$ **such that** $proposal\_number = active\_proposal\_number_i$:
    $ack\_count_i \leftarrow ack\_count_i + 1$

upon reception of $\langle \textsc{nack}$, Integer $proposal\_number$, Set $value \rangle$ **such that** $proposal\_number = active\_proposal\_number_i$:
    $proposed\_value \leftarrow proposed\_value \cup value$
    $nack\_count_i \leftarrow nack\_count_i + 1$

upon $nack\_count_i > 0$ and $ack\_count_i + nack\_count_i \geq f + 1$ and $active_i = true$:
    $active\_proposal\_number_i \leftarrow active\_proposal\_number_i + 1$
    $ack\_count_i \leftarrow 0$
    $nack\_count_i \leftarrow 0$
    trigger *beb*.broadcast$(\langle \textsc{proposal}, proposed_value_i, active_proposal_number_i \rangle)$

upon $ack\_count_i \geq f + 1$ and $active_i = true$:
    trigger decide$(proposed_value_i)$
    $active_i \leftarrow false$

Algorithm 2 Lattice Agreement Algorithm: Pseudocode of acceptor P_i

upon init:
    Set $accepted\_value_i = \bot$

upon reception of $\langle \textsc{proposal}$, Set $proposed\_value$, Integer $proposal\_number \rangle$ from proposer P_j **such that** $accepted\_value_i \subseteq proposed\_value$:
    $accepted\_value_i \leftarrow proposed\_value$
    send $\langle \textsc{ack}, proposal\_number \rangle$ to P_j

upon reception of $\langle \textsc{proposal}$, Set $proposed\_value$, Integer $proposal\_number \rangle$ from proposer P_j **such that** $accepted\_value_i \not\subseteq proposed\_value$:
    $accepted\_value_i \leftarrow accepted\_value_i \cup proposed\_value$
    send $\langle \textsc{nack}, proposal\_number, accepted\_value_i \rangle$ to P_j