General Byzantine consensus (cont'd)

Reference: Attiya and Welch, Section 5.2.

Lemma If process k is correct, all correct processes have the same preference at end of round k

Proof Suppose some correct process P_i has a different preference from P_k at the end of phase k. Then P_i must have received $> f + \frac{n}{2}$ copies of some value x in first round of phase k. At least $\frac{n}{2} + 1$ copies came from correct processes, and these $\frac{n}{2} + 1$ copies will also be sent to P_k in round 1 of phase k. Consequently x was the value P_k sent in round 2 of phase k. Thus, every correct process (including P_k) updates its preference to x at end of phase k, which contradicts the assumption.

Since we perform $f + 1$ phases, at least one of them will correspond to a correct process. At the end of that phase, all processes agree. By lemma of last day, no process will ever change its preference thereafter. So the algorithm correctly solves consensus. It uses $2f + 2$ rounds.

Summary of Results for Synchronous, Message-Passing Consensus with Byzantine Failures

Complete graphs

- $n \leq 3f$: impossible (proved last class)
- $n > 4f$: use the above algorithm (discussed in last class)
- $3f < n \leq 4f$: use more complicated algorithm (see Garay and Moses, SIAM J. Computing, 1998. Their algorithm uses only $f+1$ rounds. It works in a complete graph whenever $n > 3f$.)

General graphs

- $n \leq 3f$ or $c \leq 2f$: impossible (If $n \leq 3f$, impossibility follows from the result of last class, because removing edges from the complete graph only makes the problem *harder*. Completing the proof of impossibility for $c \leq 2f$ is homework exercise 3.)
- $n > 3f$ and $c > 2f$: possible
  Idea: simulate algorithm for complete graph as described below. See figure 1.

  For any 2 processes P and Q, there are $2f + 1$ disjoint paths from P to Q in general graph. (The existence of these paths follows from the fact that graph has connectivity $\geq 2f + 1$.) If P wants to send a message to Q, it sends copies of the message along all of the paths. If Q gets $\geq f + 1$ copies of message, then at least 1 is reliable. So it accepts the message. If it gets f or fewer copies, then the message might be a fake message, generated by Byzantine processes on f of the paths, so Q does not accept it.

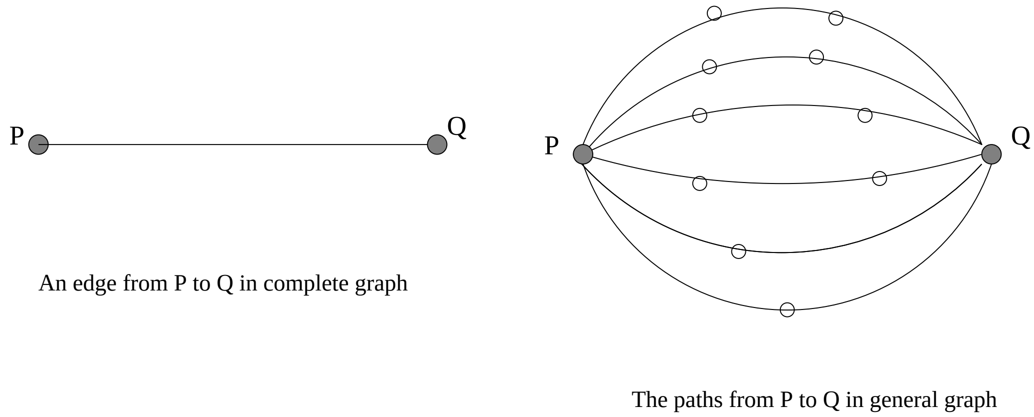

Figure 1: complete graph vs. general graph

Synchronous, Message-Passing Consensus with Halting Failures

References: [AW98], Section 5.1 and [Lyn96], Section 6.7.

In each step each process can send a message to every other process. In the step where a process dies, it can send out message to some subset of processes. i.e., a process can die "in middle of" a round.

In a *complete network*, we can solve consensus for f failures in $f+1$ rounds for any f.

We use the following definition of the consensus problem for the situation of halting failures.

- Agreement: All outputs produced are the same.
- Validity: If all processes have the same input x, all outputs produced are equal to x.

Algorithm An algorithm that works in $f+1$ rounds

Each process maintains a set of the values it knows to exist in the system; Initially the process has only seen its own input. In each step, it sends all the values it has seen so far to everybody, and updates its own set by joining it with the new values received from other processes. After $f+1$ rounds, it ouuputs the minimum value in its set.

Lemma Every nonfaulty process has seen same set of values at the end of round $f+1$

Proof Suppose P_i, P_j are nonfaulty processes. And P_i has seen value x, whereas P_j has not at the end of algorithm. Value x has to be traced from some process with input x to P_i through a chain of messages.
case I: P_i receive x in the last round. The process that sent message with value x to P_i at round $f+1$ must live and will send message to P_j as well. Because each of the previous f processes on the chain must die immediately after sending message to the next one on the chain, otherwise it would have sent the message to P_j directly.
case II: P_i receive x before last round, then in the next round $\leq f+1$, P_i sends x to P_j.

Theorem

Let $n \geq f+2$, consensus in complete, synchronous network with f failures takes at least $f+1$ rounds in the worst case.

Reminder

Proving lower bound $\Omega(f(n))$ in the worst case complexity of the problem means $\forall$ algorithm, $\exists c, \forall$ large n, $\exists$ execution for input of size n where time $\geq c \cdot f(n)$.

Define

Let α, β be executions of an algorithm.

$\alpha \overset{P}{\sim} \beta$ means α and β are indistinguishable to process P, i.e., P gets same messages at the same times in both executions.

$\alpha \sim \beta$ if $\alpha \overset{P}{\sim} \beta$ for some correct process P

$\alpha \approx \beta$: $\approx$ is the transitive closure of $\sim$. I.e. $\alpha \approx \beta$ iff $\exists r_1..r_k$ such that $\alpha \sim r_1 \sim r_2 \sim r_3 ... \sim r_k \sim \beta$.

Lemma If α, β are executions of a consensus algorithm and $\alpha \approx \beta$ then $output(\alpha) = output(\beta)$.

Proof If $\alpha \approx \beta, \exists r_1, r_2, ... r_k$ such that $\alpha = r_1 \sim r_2 \sim ... \sim r_k = \beta$. We have $output(r_i) = output(r_{i+1})$, because some process cannot distinguish the two executions so it must ouput the same value in both.

The single failure case

Before presenting the proof of the lower bound for the general class, we look at teh special case $f = 1$, $n = 3$, and prove ≥ 2 rounds required.

Suppose there is an algorithm that runs in 1 round.

Consider 4 executions with no failures, with inputs as shown in figure 2. By validity, $output(\alpha_0) = 0$ and $output(\alpha_3) = 1$.

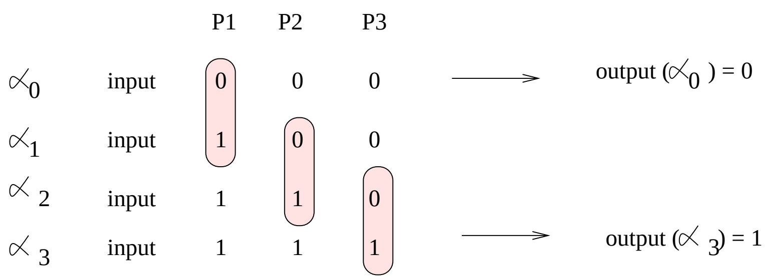

Figure 2: executions with no failures

Goal: show $\alpha_{i-1} \approx \alpha_i$ if the algorithm has only 1 round. This would mean that $\alpha_0 \approx \alpha_3$, contradicting the previous lemma.

The only difference between the inputs of α_{i-1} and α_i is the input for process P_i. We want to cover up the evidence of this difference by killing process P_i. However, if we just kill the process before it sends any messages, all other processes will notice that P_i has died. So we create some intermediate executions which differ from each other only by the removal of a single message, as shown in figure 3.

We have $\alpha_{i-1} \overset{Q}{\sim} \beta_{i-1} \overset{R}{\sim} \beta_i \overset{Q}{\sim} \alpha_i$, thus $\alpha_{i-1} \approx \alpha_i$.

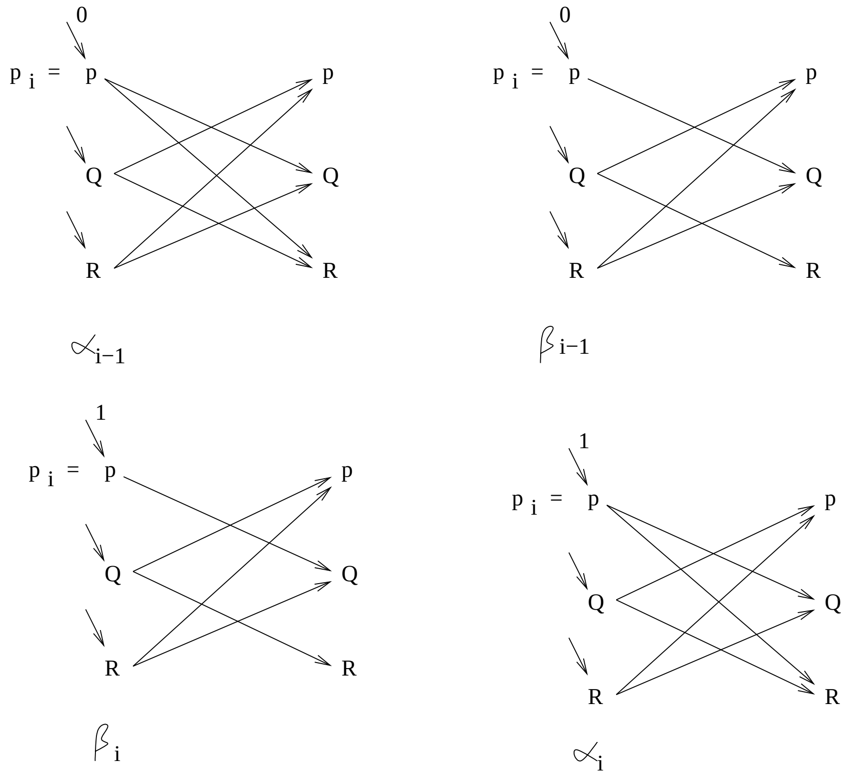

Figure 3: show $\alpha_{i-1} \overset{Q}{\sim} \beta_{i-1} \overset{R}{\sim} \beta_i \overset{Q}{\sim} \alpha_i$