Principles of Cryptocurrency Design

Appunti ed Esercizi

Francesco Pasquale

19 marzo 2026

In questi appunti completiamo la dimostrazione del seguente teorema, iniziata nella scorsa lezione.

Teorema 1 (Fischer, Lynch, Paterson [4])**.** Nel modello asincrono nessun protocollo deterministico per *Byzantine Agreement* che termina in un tempo finito può soddisfare *consistency* e *validity*, se c'è almeno un nodo corrotto.

1 Dimostrazione del teorema FLP

Si vedano gli appunti del 16 marzo per le definizioni di *Byzantine Agreement* e modello asincrono e per alcune proprietà di quest'ultimo.

Si ricordi che, dato un protocollo deterministico Π per *Byzantine agreement*, una configurazione $C = (S, M)$ e uno *schedule* $\sigma = (m_1, \ldots, m_k)$, diciamo che σ è *applicabile* a C se $m_1 \in M$ e se per ogni $i = 2, \ldots, k$, il messaggio m_i appartiene alla *message pool* della configurazione $\sigma_{[1:i-1]}(C)$, dove con $\sigma_{[1:i-1]}$ intendiamo lo *schedule* formato dai primi $i-1$ messaggi dello *schedule* σ.

Si ricordi inoltre che una configurazione C si dice 0-*valente* (rispettivamente 1-*valente*) per il protocollo Π se, qualunque cosa faccia l'avversario, a partire dalla configurazione C il protocollo fa in modo che tutti i nodi onesti danno in output 0 (rispettivamente 1). Altrimenti diciamo che una configurazione C è *bivalente*.

Nella lezione del 16 marzo abbiamo dimostrato il lemma seguente, che dice che esiste sempre una configurazione iniziale bivalente.

Lemma 2. Se Π_{BA} è un protocollo deterministico per *Byzantine Agreement* che termina e soddisfa *validity* e *consistency* allora esiste un'assegnazione degli input ai nodi tale che la configurazione iniziale è bivalente.

Nel seguito di queste note diamo una dimostrazione del secondo lemma chiave in [4].

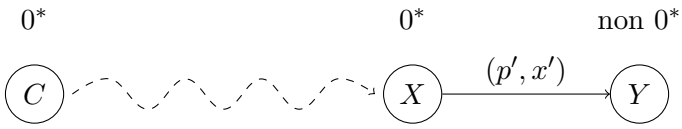
Lemma 3. Se Π_{BA} è un protocollo deterministico per *Byzantine agreement* che termina e soddisfa *validity* e *consistency*, $C = (S, M)$ è una configurazione bivalente per Π_{BA} e $m = (p, x)$ è un messaggio in M, allora esiste uno *schedule* $\sigma = (m_1 = (p_1, x_1), \ldots, m_k = (p_k, x_k))$ applicabile a C tale che

1. $m_k = m$;
2. $\sigma(C)$ è una configurazione bivalente per Π_{BA}.

Dimostrazione. Se $m(C)$ è bivalente, il lemma è dimostrato. Altrimenti $m(C)$ è 0-valente o 1-valente. Supponiamo senza perdita di generalità che $m(C)$ sia 0-valente; il caso in cui $m(C)$ è 1-valente è speculare.

Data una configurazione bivalente $\hat{C}$ per Π_{BA}, diciamo che $\hat{C}$ è di tipo 0^* per il nostro messaggio m se $m(\hat{C})$ è 0-valente. Analogamente diciamo che $\hat{C}$ è di tipo 1^* o bi*, se $m(\hat{C})$ è 1-valente o bivalente, rispettivamente.

Siccome ci siamo posti, senza perdita di generalità, nel caso in cui $m(C)$ è 0-valente, con la terminologia appena definita possiamo dire che C è una configurazione bivalente di tipo 0^*. Consideriamo tutte le configurazioni raggiungibili applicando a C degli *schedule* che non contengono il nostro messaggio $m = (p, x)$. Osservate che fra queste configurazioni ce ne deve essere qualcuna che non è di tipo 0^*, perché se fossero tutte di tipo 0^* allora la configurazione C non potrebbe che essere 0-valente, invece è bivalente per ipotesi. Indichiamo con Y la prima configurazione non 0^* raggiungibile applicando a C uno *schedule* che non contiene il messaggio m, indichiamo con X la configurazione di tipo 0^* che la precede e con $m' = (p', x')$ il messaggio dello *schedule* che secondo il protocollo porta dalla configurazione X alla configurazione Y



Il nostro messaggio $m = (p, x)$ è applicabile alla configurazione Y, perché stava nella *message pool* della configurazione C e non è mai stato consegnato, quindi sta anche nella message pool di Y. Sia allora $Z = m(Y)$ la configurazione che si ottiene applicando m a Y. Mostreremo che Z è la configurazione bivalente che stiamo cercando, concludendo così la dimostrazione del lemma.

La configurazione Z non può essere 0-valente, altrimenti Y sarebbe di tipo 0^*. Ora mostriamo che Z non può essere neanche 1-valente esibendo uno *schedule* applicabile a Z che conduce a una configurazione in cui tutti i nodi onesti danno in output 0.

Prima di tutto osserviamo che se Z non è 0-valente allora deve essere $p = p'$. Infatti, se fosse $p \neq p'$, la configurazione Z, che si ottiene da X consegnando prima (p', x') e poi (p, x), sarebbe la stessa configurazione che si ottiene da X consegnando prima (p, x) e poi (p', x') (vedi l'Esercizio 4 degli appunti del 16 marzo), ma quest'ultima configurazione è sicuramente 0-valente, perché X è di tipo 0^*.

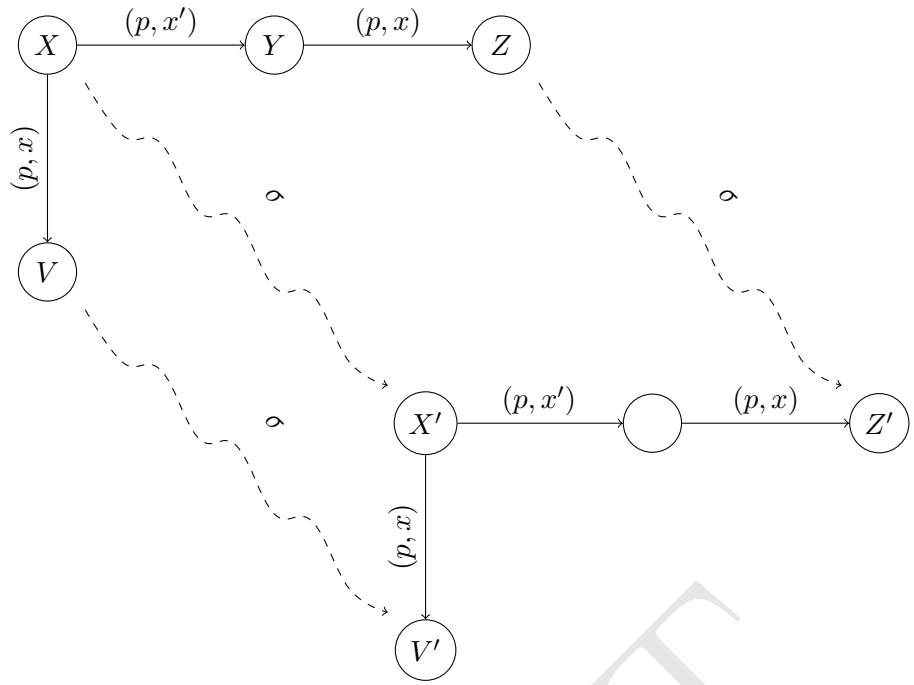
Siccome $p = p'$ e siccome per ipotesi Π_{BA} termina anche in presenza di un nodo corrotto, deve esistere uno *schedule* σ applicabile a X tale che σ non contiene messaggi per il nodo p e $X' := \sigma(X)$ è una configurazione in cui tutti i nodi diversi da p che seguono il protocollo hanno deciso il loro output. Infatti, se un tale *schedule* non ci fosse, nel caso in cui p è un nodo corrotto che non fa nulla ogni volta che riceve un messaggio, nessuno *schedule* consentirebbe agli altri $n - 1$ nodi di terminare.

Lo *schedule* σ non contiene messaggi per il nodo p, quindi è applicabile anche alla configurazione $m(X)$ e, poiché $p' = p$, anche alla configurazione Z:

1. Siccome $\sigma(X)$ è una configurazione in cui tutti i nodi diversi da p devono aver deciso il loro output, anche $m(\sigma(X))$ lo è. Ma quindi l'output deciso dai nodi deve essere 0, perché $m(\sigma(X)) = \sigma(m(X))$ e $m(X)$ è una configurazione 0-valente dato che X è di tipo 0^*.

2. La configurazione $\sigma(Z) = \sigma(m(m'(X))) = m(m'(\sigma(X)))$ e siccome $\sigma(X)$ è una configurazione in cui tutti i nodi diversi da p hanno deciso che il loro output è 0, anche $\sigma(Z)$ lo è.

Quindi abbiamo trovato uno *schedule* σ applicabile a Z tale che i nodi danno in output 0. Quindi Z non può essere neanche 1-valente e pertanto deve essere bivalente.



□

Usando il Lemma 2 e il Lemma 3 possiamo infine dare una dimostrazione del Teorema 1.

Dimostrazione. Supponiamo per assurdo che esista un protocollo deterministico Π_{BA} che termina e soddisfa *validity* e *consistency* nel modello asincrono anche in presenza di un nodo corrotto. Per il Lemma 2 deve esistere una configurazione iniziale bivalente $C_0 = (S_0, M_0)$ per Π_{BA}. Per ogni $i \geqslant 1$, sia m_i il messaggio "più vecchio" presente in M_{i-1} (rompendo la simmeria arbitrariamente in caso di più messaggi), sia σ_i lo *schedule* che termina con m_i la cui esistenza è garanita dal Lemma 3 e sia $C_i = \sigma_i(C_{i-1})$ la configurazione bivalente risultante.

Lo *schedule* $(\sigma_1, \sigma_2, \sigma_3, \dots)$ è quindi uno *schedule* infinito che garantisce che ogni messaggio inserito nella *message pool* viene prima o poi consegnato e fa in modo che il protocollo rimanga sempre in configurazioni bivalenti. □

Il Teorema 1 afferma che nel modello asincrono non può esistere nessun protocollo *deterministico* per *Byzantine Agreement* in presenza di qualche nodo corrotto. Questo non esclude la possibilità che sia possibile progettare protocolli *probabilistici*, infatti ce ne sono diversi e nella prossima lezione ne analizzeremo uno. Chi volesse approfondire questi argomento può vedere, per esempio, [1, 5, 3, 2].

Riferimenti bibliografici

[1] Michael Ben-Or. Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, 1983. `https://dl.acm.org/doi/pdf/10.1145/800221.806707`.

[2] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 123–132, 2000. `https://dl.acm.org/doi/pdf/10.1145/343477.343531`.

[3] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 148–161, 1988. https://dl.acm.org/doi/pdf/10.1145/62212.62225.

[4] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985. https://dl.acm.org/doi/pdf/10.1145/3149.214121.

[5] Michael O. Rabin. Randomized Byzantine Generals. In *24th annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409. IEEE, 1983. https://ieeexplore.ieee.org/iel5/4568048/4568049/04568104.pdf.