**Consenso distribuito con errori di collegamento**

Studieremo il problema del **raggiungimento del consenso** in una rete distribuita. In questo problema ognuno dei processi della rete inizia con un valore iniziale di un particolare tipo e si suppone che l’output sia un valore dello stesso tipo.

Gli output devono essere gli stessi (i processi devono **concordare**) mentre gli inputs possono essere arbitrari. Generalmente c’è una **condizione di validità** che descrive i valori di output permessi per ogni schema di inputs.

Quando non ci sono errori nelle componenti del sistema, i problemi di consenso sono solitamente di facile risoluzione utilizzando un semplice scambio di messaggio e per rendere le cose più interessanti questi problemi sono considerati in contesti che includono fallimenti. In questo capitolo considereremo i problemi di consenso con presenza di errori di comunicazione.

Troviamo i problemi di consenso in molte applicazioni di calcolo distribuito, ad esempio i processi possono tentare raggiungere un accordo sull’opportunità di impegnarsi o interrompere i risultati di una transazione di un database distribuito oppure possono provare a concordare una stima dell’altitudine di un aereo basata sulle letture di più altimetri o ancora possono tentare di accordarsi sull'opportunità di classificare un sistema componente come difettoso, visti i risultati di test diagnostici separati eseguiti da processi separati.

Il particolare problema che presenteremo qui è il **problema dell’attacco coordinato**, un problema fondamentale di consenso in un contesta nel quale i messaggi potrebbero andare persi. Cominceremo presentando l’impossibilità di un risultato per la versione deterministica e poi esploreremo la possibilità di una soluzione randomizzata. Mostreremo che il problema può essere risolto da un algoritmo randomizzato con una certa (sostanziosa) probabilità di errore e che questa probabilità di errore sia inevitabile.

**Attacco coordinato – versione deterministica**

Descriviamo brevemente il problema:

*Diversi generali stanno pianificando un attacco coordinato da diverse direzioni contro un obiettivo comune e sanno che l’unico modo per avere successo è che tutti i generali attacchino allo stesso momento, altrimenti la sconfitta sarà certa. Ogni generale ha un parere iniziale sul fatto che il proprio esercito sia pronto ad attaccare o meno.*

*I generali si trovano in luoghi diversi e quelli nelle vicinanze possono comunicare utilizzando messaggeri che viaggiano a piedi, i quali possono perdersi o essere catturati dal nemico e in questi i casi il messaggio andrà perduto. Utilizzando solo questo inaffidabile mezzo di comunicazione, i generali devono mettersi d’accordo sul fatto di attaccare o meno, inoltre, loro dovrebbero cercare di attaccare se possibile.*

*(Supponiamo che il “grafo di comunicazione” dei generali sia connesso e non orientato e che tutti i generali conoscano il grafo. Assumiamo anche che ci sia un limite superiore noto sul tempo che ci vuole ad un messaggero per consegnare con successo un messaggio.*

Se tutti i messaggeri sono affidabili, allora tutti i generali possono inviare messaggeri a tutti gli altri generali (possibilmente in più salti), dicendo se loro sono disposti o meno ad attaccare. Dopo un numero di “giri” pari al diametro del “grafico di comunicazione” tutti i generali avranno tutte queste informazioni. In seguito, tutti possono applicare una regola concordata per prendere la stessa decisione sul fatto di attaccare, ad esempio potranno attaccare solo se tutti i generali lo vogliono.

Nel modello nel quale i messaggi possono andare perduti, questo semplice algoritmo non funziona ed inoltre non è un problema solo di questo particolare algoritmo, infatti, non esiste un algoritmo che risolve sempre questo problema.

Il vero problema informatico dietro questa descrizione è quello di un impegno per i database distribuiti, il quale implica una raccolta di processi che hanno partecipato all’elaborazione di una transazione di database. Dopo questa elaborazione, ogni processo arriva ad una opinione iniziale sul fatto che la transazione debba essere impegnata (i suoi risultati debbano essere resi permanenti e rilasciati per l’utilizzo di altre transazioni) o interrotta (i suoi risultati vengono scartati). Generalmente, un processo favorirà l’esecuzione della transazione se tutto il suo calcolo locale è stato completato con successo mentre la annullerà in caso contrario. I processi dovrebbero comunicare ed eventualmente concordare uno dei risultati, **commit** o **abort**, e, se possibile, il risultato dovrebbe essere commit.

Prima di provare l’impossibilità, enunciamo il problema più formalmente per rimuovere la ambiguità.

Consideriamo processi indicizzati da disposti in una rete arbitraria di grafi non orientati dove ogni processo conosce l’intero grado, compresi gli indici di ogni processo. Ogni processo inizia con un input in , dove denota **attaccare** o **commit**, mentre indica **non attaccare** o **abort**. Utilizzeremo lo stesso modello sincrono con il quale abbiamo lavorato fino ad ora con la differenza che in questo caso consentiamo che un qualsiasi numero di messaggi vada perso durante l’esecuzione. L’obiettivo è che tutti i processi alla fine producano decisioni in impostando lo stato di decisione a o .

Ci sono tre condizioni imposte alle decisioni assunte dai processi:

**Accordo**

* Non esistono due processi che decidono valori differenti

**Validità**

* Se tutti i processi iniziano con , allora è l’unico valore di decisione possibile
* Se tutti i processi iniziano con e tutti i messaggi vengono consegnati, allora è l’unico valore di decisione possibile

**Terminazione**

* Tutti i processi alla fine decidono

Le condizioni di accordo e di validità sono naturali, mentre quello di validità è solo una possibilità su diverse alternative. Le condizioni di validità esprimono la nozione che il valore deciso dovrebbe essere “ragionevole”, ad esempio, in questo caso il protocollo banale che decide sempre è escluso dalla seconda condizione di validità. Questo particolare requisito di validità che abbiamo affermato è abbastanza debole, infatti, se ad esempio un processo inizia con , questo può decidere solamente se tutti gli altri processi iniziano con e se anche un solo messaggio viene perso allora l’algoritmo può decidere .

Questa formulazione debole è appropriata perché il nostro obiettivo principale è quello di dimostrare l’impossibilità di un risultato in un qualsiasi grafo con due o più nodi. Dimostreremo l’impossibilità per il caso particolare di due nodi connessi tra loro.

|  |
| --- |
| ***Teorema 5.1***  *Sia il grafo composto dai nodi e connessi da un singolo edge. Non esiste un algoritmo che risolve il problema dell’attacco coordinato su* |

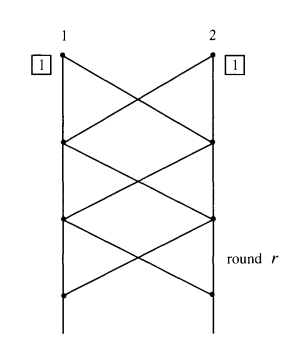
**Dimostrazione**

Per Assurdo.

Supponiamo che esista una soluzione e la chiamiamo . Assumiamo che, per ogni processo, ci sia un solo stato iniziale contente ciascun valore di input e questo implica che il sistema ha esattamente un’esecuzione per un’assegnazione fissata di input e un pattern fisso di risultati riusciti. Supponiamo inoltre che entrambi i processi inviino messaggi ad ogni turno in dal momento che possiamo sempre costringerli ad inviare messaggi fittizi.

Sia l’esecuzione nella quale entrambi i processi iniziano con il valore e tutti i messaggi sono consegnati. Per la condizione di terminazione, entrambi i processi dovranno decidere alla fine e, per la seconda condizione di validità, entrambi dovranno decidere . Supponiamo che entrambi decidano in turni.

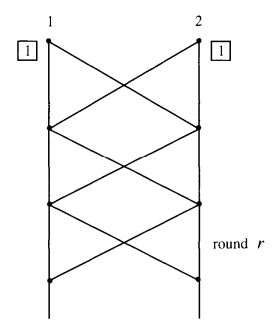
Sia uguale ad con la differenza che tutti i messaggi dopo i primi rounds vadano persi. Anche qui i processi decidono in rounds. Il pattern di comunicazione di è rappresentato dalla seguente figura:



Gli edges rappresentano i messaggi che sono stati consegnati, mentre quelli inviati ma non ricevuti semplicemente non sono disegnati.

Iniziando da , costruiamo una serie di esecuzioni dove ognuna di esse sia indistinguibile da quella precedente da parte di uno dei processi, segue che tutte queste esecuzioni avranno lo stesso valore di decisione.

Sia un’esecuzione uguale ad con la differenza che l’ultimo (turno ) messaggio dal processo al processo non sia stato consegnato come mostra la figura seguente:



Poi, sebbene il processo potrebbe assumere stati differenti dopo il round nelle esecuzioni e , questa differenza non verrà mai comunicata al processo e pertanto per il processo e per questo motivo, dato che il processo decide in , allora deciderà anche in . Per le condizioni di terminazione e di accordo, anche il processo dovrà scegliere in .

Sia un’esecuzione uguale ad , tranne per il fatto che l’ultimo messaggio dal processo al processo è perduto; quindi, per il processo pertanto il processo deciderà in e, per le condizioni di terminazione e di accordo, farà lo stesso il processo .

Continuando in questo modo, rimuovendo in modo alternato l’ultimo messaggio dal processo e dal processo , raggiungiamo infine un’esecuzione nella quale entrambi i processi iniziano con e nessun messaggio è consegnato. Per lo stesso ragionamento di prima, entrambi i processi sono forzati a decidere in questo caso.

Consideriamo ora l’esecuzione nella quale il processo inizia con ed il processo inizia con e nessun messaggio è consegnato. Abbiamo che per il processo quindi il processo decide sempre in e, per le condizioni di accordo e terminazione, fa lo stesso il processo . Ma per il processo si ha che , dove è l’esecuzione nella quale entrambi i processi partono con e nessun messaggio è consegnato. Il processo decide in e questo è assurdo perché contraddiciamo la prima condizione di validità, la quale vuole che entrambi i processi decidano in .

Il teorema descrive un limite fondamentale sulle capacità delle reti distribuite e suggerisce che c’è poco da fare per risolvere i problemi di consenso come questo in caso di comunicazione inaffidabile. Tuttavia, alcune versioni di questo problema devono essere risolte in sistemi reali e per far fronte alle limitazioni del teorema è necessario rafforzare il modello o allentare i requisiti del problema.

Un approccio consiste nel fare alcune ipotesi probabilistiche sulla perdita di messaggi, mantenendo i processi deterministici. Dobbiamo consentire qualche possibilità di violazione delle condizioni di accordo e/o di validità. Un secondo approccio consiste nel consentire ai processi di utilizzare la randomizzazione e consentire la possibilità di violazione delle condizioni di accordo e/o di validità, questo approccio lo vedremo nella prossima sezione.

**Attacco coordinato – versione randomizzata**

5.2 Problema dell’attacco coordinato, versione randomizzata

Consideriamo il problema dell’attacco coordinato con n nodi inseriti in un grafo bidirezionale ove ogni nodo conosce la posizione ed il numero degli altri.

Al solito ogni processo nasce con un input {0,1} (attaccare/non attaccare) e tutti quanti si devono mettere d’accordo entro un numero *r* >= 1 di round.

Però adesso introduciamo la possibilità di un disaccordo, accettiamo che ci sia una probabilità ϵ che i processi siano in disaccordo.

ϵ indica quanto è probabile che i processi siano in disaccordo.

5.2.1 Formalizzazione

La difficoltà di formalizzare questo modello sta nel fatto che il risultato non dipende solo dal fatto che i messaggi sono persi, ma da quali messaggi sono persi.

Quindi introduciamo il concetto di un "avversario" che causa la perdita di messaggi in modo da rendere le cose il più difficile possibile per il nostro algoritmo.

Definiamo un *communication pattern* come un sottoinsieme ɣ:

 ɣ sottoinsieme di {(*i, j, k* ) t.c. ( *i , j*) sono nodi del grafo, e 1 ≤ k, ove *k* è un turno}.

Diciamo che il pattern ɣ è *“buono”* se k ≤ *r*, cioè il sottoinsieme di tutti i “messaggi” inviati da tutti i nodi del grafo dal round 1 al round *r.*

Dire (i,j,k) significa dire che è stato inviato un messaggio da *i* a *j* durante il round *k* ed il messaggio è arrivato senza perdersi.

Un possibile avversario è una combinazione di: 1 l’assegnamento di un input a tutti i processi, 2 un pattern “*buono*”. Ricordiamo che stiamo cercando l’avversario che rende le cose più difficili possibile.

Per ogni possibile avversario B possiamo indicare la probabilità che l’avversario induce come PB. Dato un avversario B la probabilità che anche solo uno dei nodi sia in disaccordo sara PB

Il nostro scopo è cercare l’avversario più sconveniente, cioè l’avversario B per cui PB = ϵ

Trovato il valore di ϵ dobbiamo dimostrare che non esiste alcun avversario A t.c. PA > ϵ

5.2.2 Algoritmo ingenuo

Mostriamo un algoritmo che riesce ad ottenere un ϵ = 1/r

Per prima cosa definiamo il una caratteristica del pattern ɣ, un ordinamento parziale ≤ɣ composto dalle coppie (*i,k*) che indicano un processo *i*  ad un dato tempo *k*.

≤ɣ rappresenta la “quantità di informazione” ricevuta dai nodi (quante informazioni hanno sugli altri nodi, che poi è proporzionale al numero di messaggi ricevuti).

≤ɣ può essere visto come “ha meno conoscenza”

1. ( i, k) ≤ɣ ( i, k’) per ogni i se 1 ≤ i ≤ n e 0 ≤ k ≤ k’. In pratica se *i* è un nodo del grafo la quantità di informazione aumenta o rimane stabile con il tempo, non diminuisce. un nodo *i* riceve o invia messaggi, quindi l’informazione aumenta.
2. se ( i, j, k) appartiene a ɣ, ( i, k-1) ≤ɣ (j, k). Presi 2 nodi i e j, che possiamo vedere come mittente e destinatario di un messaggio, la quantità di informazioni da i al tempo k-1 è minore o uguale alla quantità di informazioni di j al tempo k. i può aver inviato un messaggio a j incrementando la sua quantità di informazioni. l’informazione fluisce da mittente a destinatario.  
   L’invio di un messaggio da mittente a destinatario incrementa l’informazione del destinatario
3. se (i,k) ≤ɣ (i’, k’) e (i’,k’) ≤ɣ (i’’,k’’) allora (i,k) ≤ɣ (i’’,k’’). “≤ɣ” è una operazione transitiva

definito ≤ɣ possiamo anche formalizzare il livello di informazione di un nodo *i*  al tempo *k*, chiamiamolo *level*ɣ(i,k).

1. se k = 0.  
   levelɣ(i,k) = 0, il nodo non può aver ricevuto alcuna informazione
2. se k >0 ed esiste un j != i tale che (j,0) !≤ɣ (i,k)

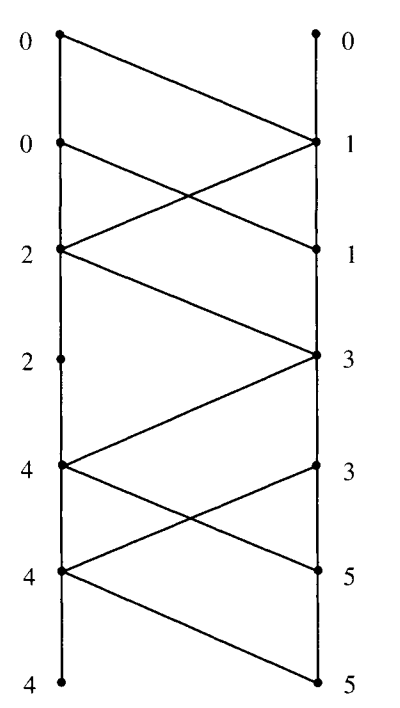
levelɣ (i,k) = 0. la situazione (j,0) !≤ɣ (i,k) si verifica solamente quando i round passano ma *i* non riceve alcun messaggio, perciò se *i* non riceve messaggi il suo livello non aumenta.

1. se k >0 e (j,0) ≤ɣ (i,k) per tutti i *j* != *i*  
   levelɣ(i,k) = 1 + min{ max{levelɣ(j,k), considerando tutti i k} , considerando tutti i j != i}. In pratica ogni processo non può passare ad un livello successivo finchè non ha saputo che tutti gli altri processi hanno raggiunto il livello attuale.  
   Ogni processo parte a 0, quando tutti gli altri processi gli dicono di essere al livello 0 passa al livello uno, quando riceve il messaggio da tutti gli altri che sono a livello 1 passa a livello due eccetera.

Il lemma 5.2 garantisce proprio questo.

Dati i e j con i != j in un tempo k tc 0 ≤ k ≤ r allora levelɣ(i,k) si distacca al massimo di 1 da levelɣ(j,k).

Guardando l’immagine 5.2.1 che rappresenta i livelli in relazioni ai messaggi inviati e ricevuti notiamo che se tutti i messaggi fossero ricevuti levelɣ di ogni nodo sarebbe uguale al numero di round. Lemma 5.3



**Algoritmo informale**

Possiamo finalmente scrivere l’algoritmo, la cui idea é:

ogni processo tiene traccia del suo *livello* (e di quello degli altri), e di un valore “key”.

*key* è un intero compreso tra 1 ed *r,* deciso dal processo n°1, questo valore *key* è presente all’interno di ogni messaggio inviato, se il nodo che invia il messaggio lo sa (piggybacked).

Vengono inviati anche tutti i valori di decisione iniziali di ogni nodo, piggybacked (non so come esprimere il concetto di piggybacked, la prof lo spiega bene in una delle prime lezioni sugli algoritmi distribuiti). In pratica ogni nodo ha una lista di valori iniziali, un campo per ogni altro nodo, quando riceve un messaggio che contiene un valore di cui lui non è a conoscenza modifica la sua lista e nei prossimi messaggi che invia inserirà la lista aggiornata, causando un effetto cascata.

Dopo *r*  round ogni processo decide 1 se e solo se il suo livello è >= al valore key e tutti gli initial values dei nodi sono = 1 naturalmente lui può fare considerazioni solo sulla sua “lista”, gli initial values che gli sono arrivati

**Algoritmo formale (Automa)**

Questa parte non va “capita” è un formalizzazione. per capire questa sezione bisogna assolutamente aver visto la seconda lezione della prof sugli algoritmi distribuiti. Sicuramente va inserita nella presentazione.

statesi:

int rounds,  initially 0

decision ∈ {unknown, 0, 1}, initially unknown

key ∈ [1, r] U undefined, initially undefined # a meno che i non sia il primo processo, in tal caso randi

for every j, 1 < j ≤ n: #n numero processi

val(j) ∈ {0, 1, undefined}; initially val(i) is i's initial value and

val(j) = undefined for all j != i

level(j) ∈ [-1, r]; initially level(i)= 0 and level(j)= -1 for all j != i

ricordo che key è una variabile, val e level sono due vettori. level è un vettore contenente il massimo livello conosciuto di ogni nodo (compreso il nodo stesso i).

val un vettore contenente la decisione iniziale di ogni nodo (se conosciuta)

randi:

if i = 1 and rounds = 0 then key = random [tra 1 ed r]

msgsi:

send (L, V, key) to all j, where L is the level vector and V is the val vector

transi:

rounds := rounds + 1

let (Lj, Vj , Kj ) be the message from j, for each j from which a message arrives

if, for some j, Kj != undefined then *key* = Kj #se la chiave te l’ha detta qualcuno che la sa te la salvi

for all j != i do

if, for some i’, Vi,(j) != undefined then val(j) := Vi,(j)

if, for some i’, Li( j ) > level(j) then level(j):= max, {Li,(j)}

level(i) := 1 + min {level(j): j != i}

if rounds = r then

if key != undefined and level(i) >= key and val(j) = 1 for all then

decision := 1

else decision := 0

Teorema 5.4

L'Algoritmo ingenuo del RandomAttack risolve la versione randomizzata del problema dell’attacco coordinato (versione randomica)  con un ϵ = 1/r ove r è il numero di round

Dimostrazione:

Per dimostrare la correttezza di questo algoritmo bisogna dimostrare che l’algoritmo calcola correttamenti i livelli: cioè ad ogni round k, per il nodo *i* il valore di *level(i)* ( cioè il livello del nodo *i*  stesso) è uguale a levelɣ(i,k).

Dato il lemma 5.2 possiamo affermare che se *level*(i) >= 1 sicuramente *key* è definita e contiene il valore deciso dal processo 1 e il vettore *val* è riempito dei valori iniziali di tutti i nodi. poichè se *i* è potuto “salire” a livello 1 allora deve aver ricevuto i dati di tutti gli altri nodi, per quanto detto al punto 3 della definizione formale di levelɣ(i,k).

Siamo sicuri che il processo termina, in quanto dopo *r* round la decisione viene sicuramente presa, la dimostrazione di Terminazione è ovvia

supponiamo che tutti i processi partano con valore 1 e tutti i messaggi vengano trasmessi, il lemma 5.3 garantisce che level(i) = *r*. dato che *r* >= 1 vale il discorso fatto con di *level*(i) >= 1. perciò *key* è definita, *key* è ≤ di *level(i)*, *value* è definito ed è uguale 1 per tutti i nodi quindi il nodo *i* sa che tutti gli altri hanno deciso 1. tutti i nodi sono d'accordo dato che 1 è l’unica decisione possibile.

Consideriamo invece un avversario B, dimostriamo che PB(almeno un processo sia in disaccordo) ≤ ϵ

Chiamiamo *li* il valore di *level(i)* nel momento in cui *i* prende la decisione (il round r).

Il Lemma 5.2 garantisce che tutti i livelli dei processi si distaccano al massimo di 1.

L’unica situazione in cui i processi possono essere in disaccordo è quando min{ *li }* < *key ≤ max{ li }* cioè quando il valore di *level(i)* non è lo stesso per tutti i processi e *key* è più grande del valore più piccolo.

Se key fosse ≤ min{ *li* } tutti sono d’accordo perchè tutti hanno superato *key.*

Se invece *key > max{ li }* si decide 0 perchè non si è raggiunto il livello necessario.

Ma i due valori discordanti di *level(i)* si distaccando di 1 per il lemma 5.2, quindi dire

min{ *li }* < *key ≤ max{ li }* è come dire *key* = max { *li* }, essendo la chiave un valore randomico tra 1 ed *r* la probabilità che ciò succeda è 1/*r*

ϵ=1/r

**Limite inferiore sul disaccordo**

Ora mostreremo che non è possibile fare molto meglio del limite descritto dal .

|  |
| --- |
| ***Teorema 5.5***  *Ogni algoritmo r-round per il problema dell’attacco coordinato randomizzato presenta una probabilità di disaccordo di al minimo* |

Per il resto di questa sezione assumeremo un particolare algoritmo che risolve il problema dell’attacco coordinato con una probabilità di disaccordo pari a in un grafo completo di nodi e dimostreremo che .

Per dimostrare questo teorema abbiamo bisogno di un’ulteriore definizione. Se è un qualsiasi avversario, il suo pattern di comunicazione e un qualsiasi processo, allora definiamo un altro avversario, .

L’avversario semplicemente “elimina” le informazioni che non ha ricevuto in .

è definito come segue:

1. Se , allora l’input di in è lo stesso che in , altrimenti è
2. Una tripla è nel pattern di comunicazione di esattamente se è nello schema di comunicazione di e

Questo vuol dire che l’avversario include tutti i messaggi che conosce in , ma non altri, e specifica che tutti gli input che non conosce in sono . Il seguente lemma dice che la versione di un avversario è sufficiente a determinare la distribuzione di probabilità degli output.

***Lemma 5.6***

*Se e sono due avversari, è un processo e , allora:*

***Lemma 5.7***

*Sia un qualsiasi avversario per il quale tutti i valori inziali sono e sia un processo qualsiasi, allora:*

**Dimostrazione lemma 5.7**

Per assurdo su .

**Passo base:** Supponiamo , definiamo e allora e quindi per il lemma 5.6:

**Equazione 5.1**

Poiché allora ci deve essere qualche processo tale che , dove è il communication pattern di . Allora l’avversario specifica un valore iniziale di per e non include messaggi con destinazione nel proprio communication pattern. Segue che è il banale avversario per tutti quelli il cui valore iniziale è e per i quali non ci sono messaggi nel communication pattern.

Sia questo banale avversario, allora , e per il lemma 5.6:

La condizione di validità implica che:

Perciò:

Ma poiché la probabilità di disaccordo è al massimo abbiamo che:

Perciò:

Il che, per l’equazione 5.1, implica che:

E pertanto il passo base è dimostrato.

**Passo induttivo:** Supponiamo e supponiamo che il lemma valga per tutti i livelli minori di , definiamo , allora il lemma 5.6 implica che:

**Equazione 5.2**

Poiché , la definizione di implica che ci deve essere qualche processo tale che e per l’ipotesi induttiva abbiamo:

Ma poiché la probabilità di disaccordo è al massimo abbiamo che:

Perciò:

Il che, per l’equazione 5.2, implica che:

E pertanto anche il passo induttivo è dimostrato.

Possiamo ora provare il teorema.

**Dimostrazione teorema 5.5**

Sia l’avversario per il quale tutti gli inputs sono e nessun messaggio è perduto.

La probabilità che tutti i processi decidano è al più la probabilità ognuno di essi decida , la quale è, per il lemma 5.7, al massimo

La condizione di validità ci dice che tutti i processi devono decidere in tutte le esecuzioni generate da questo avversario quindi la probabilità che tutti decidano deve essere esattamente e questo implica che:

Ovvero:

CVD