Reti asincrone

Problema del consenso in caso di malfunzionamenti

Borello Nazareno, Galatola Marco, Mecca Francesco

Importanza del problema

“Before this paper, it was generally assumed that a three-processor system could tolerate one faulty processor”

Marshall Pease, Robert Shostak, Leslie Lamport Journal of the Association for Computing Machinery 27 | April 1980, Vol 2

“Over the years, I often wondered whether the people who actually build airplanes know about this problem. In 1997, I received email from John Morgan who used to work at Boeing. He told me that he came across our work in 1986 and that, as a result, the people who build the passenger planes at Boeing are aware of the problem and design their systems accordingly”

2005 Edsger W. Dijkstra Prize in Distributed Computing

Impossibilità di consenso

Teorema

Non esiste un algoritmo, per il modello **broadcast asincrono con canali sicuri**, che risolva il problema del consenso e garantisca *1-failure termination*

La dimostrazione di basa su:

- il teorema di impossibilità per il modello a memoria condivisa
- l'esistenza di una trasformazione dal modello broadcast asincrono al modello con memoria condivisa

Cosa succede per il modello asincrono a rete send/receive?

Si ottiene lo stesso risultato perché si può dimostrare che i due modelli sono computazionalmente equivalenti

Possibili soluzioni

Sono state proposte varie soluzioni per risolvere il problema del consenso in presenza di malfunzionamenti, di seguito verranno trattati:

- **algoritmi randomizzati**
- **failure-detector**

Problema del consenso

Nel problema del consenso ogni processo *i* parte con un suo valore iniziale e decide un suo valore finale. Il consenso è quindi definito in termini di due primitive:

Input:

$init(v)_i$: inizializzazione di una variabile interna con il valore scelto

Output:

$decide(v)_i$: viene offerto il consenso per il valore v

$v \in V,\ 1 \leq i \leq n$

Per modellare il fallimento di un processo si ipotizza un ulteriore evento di input $stop_i$

Proprietà del consenso

Agreement: tutti i processi decidono lo stesso valore

Validity: se viene effettuata una *init* con lo stesso valore *v* su ogni processo allora *v* sarà l'unico valore possibile per l'operazione *decide*

Well Formedness: ogni sequenza di $init_i$ e $decide_i$ è un prefisso della sequenza $init(v)_i, decide(w)_i$

Terminazione

Failure-free: in ogni esecuzione fair e failure-free ogni processo effettua una *decide*

F-failure ($0 \leq f \leq n$): in ogni esecuzione fair in cui gli eventi di *init* avvengono su tutte le porte, se ci sono al massimo f eventi di *stop*, gli eventi *decide* avvengono per tutti i processi sani

Wait-free: f-failure termination con $f = n$

Algoritmi randomizzati

Alcune scelte **vengono effettuate in maniera probabilistica**, utilizzando la funzione *RANDOM(A)* che restituisce con probabilità uniforme un elemento dell'insieme A scelto casualmente

Si indeboliscono le condizioni di correttezza, la terminazione è ora probabilistica:

si richiede che processi sani eseguano l'azione $decide_i(v)$ ad un tempo t con probabilità $p(t)$ dove p è una funzione monotona, non decrescente e non limitata

Algoritmo di Ben-Or

Si richiede che : $n > 3f$ e $V = \{0,1\}$

Funzionamento:

- Ogni processo P_i ha due variabili locali x e y inizialmente nulle
- L'evento di input $init(v)_i$ fa sì che il processo P_i inizializzi la propria variabile $x = v$
- Ogni processo P_i esegue una serie di **step**, ciascuno costituito da due **fasi**. Il processo P_i inizia il round 1 dopo aver ricevuto il valore iniziale con l'evento di input $init(v)_i$

I processi continuano l'esecuzione dell'algoritmo anche dopo aver eseguito l'azione di $decide_i(v)$

Algoritmo di Ben-Or

Fase 1:

P_i invia in broadcast (*"first", s, v*), dove *v* è il valore corrente di *x*, poi aspetta di ricevere $n - f$ messaggi della forma (*"first", s*, *), se tutti questi messaggi riportano lo stesso valore *v*, allora P_i pone $y = v$, altrimenti pone $y = null$

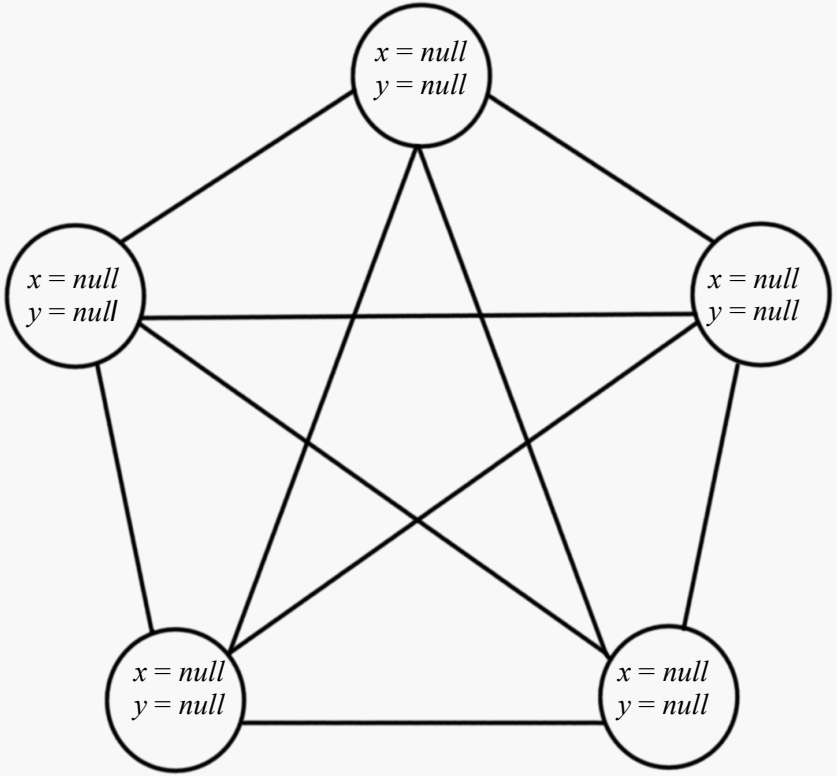
Fase 2:

P_i invia in broadcast (*"second", s ,v*), dove *v* è il valore corrente di *y* , poi aspetta di ricevere $n - f$ messaggi della forma (*"second", s,* *). Ci sono tre casi:

1. Se tutti gli $n - f$ messaggi riportano lo stesso valore $v \neq null$, allora P_i pone x= v ed esegue un'azione di $decide(v)_i$ a meno che l'abbia già fatto in precedenza
2. Se non tutti, ma almeno $n - 2f$ degli $n - f$ messaggi ricevuti riportano lo stesso valore $v \neq null$ allora P_i pone $x = v$ ma non decide (l'ipotesi di $n>3f$ implica che non ci possano essere due valori diversi di *v*)
3. Se non si verifica nessuno dei casi precedente, P_i pone $x = 0 \vee x = 1$ scegliendo a **caso** uno dei due valori con uguale probabilità

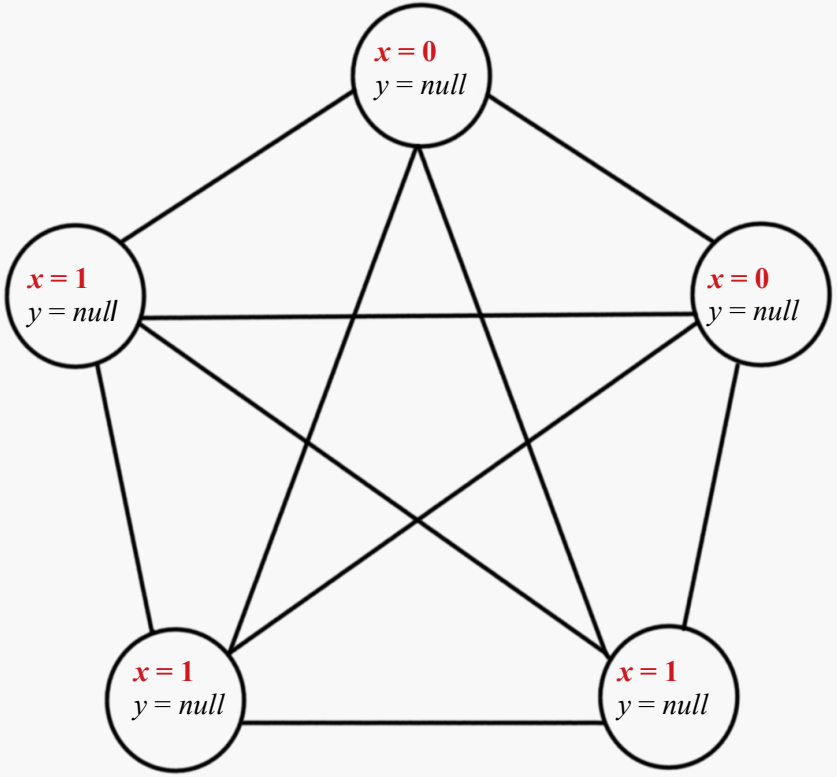
Esempio: inizio

Stato iniziale



Esempio: inizio

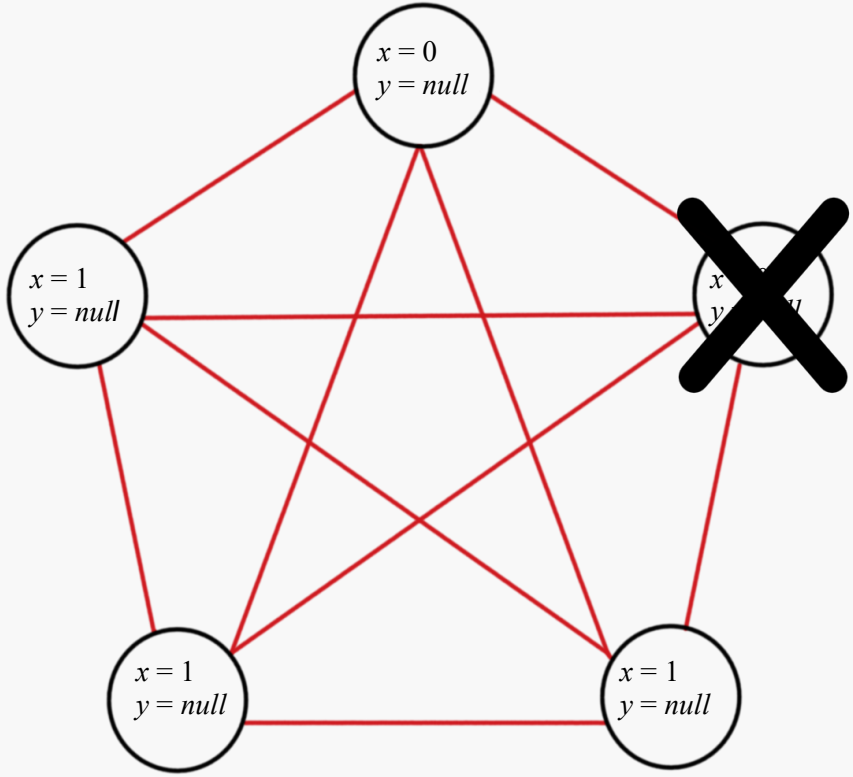
Viene eseguita ***init(v)*** su tutte le porte



Esempio: fase 1, step 1

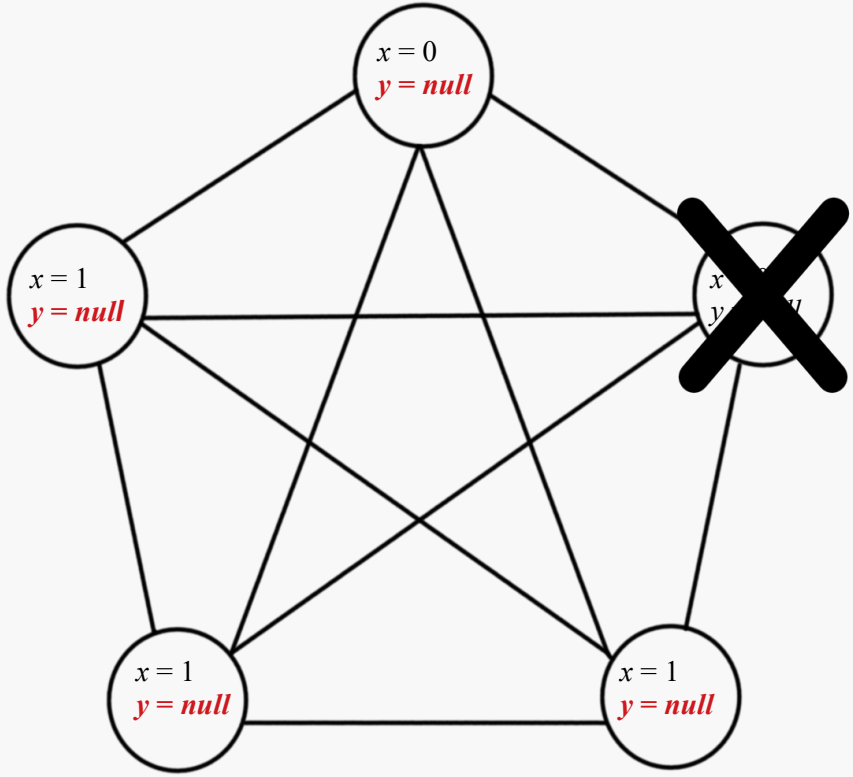
Ogni processo manda in **broadcast (*"first"*, *1*, *x*)**

Uno dei processi fallisce



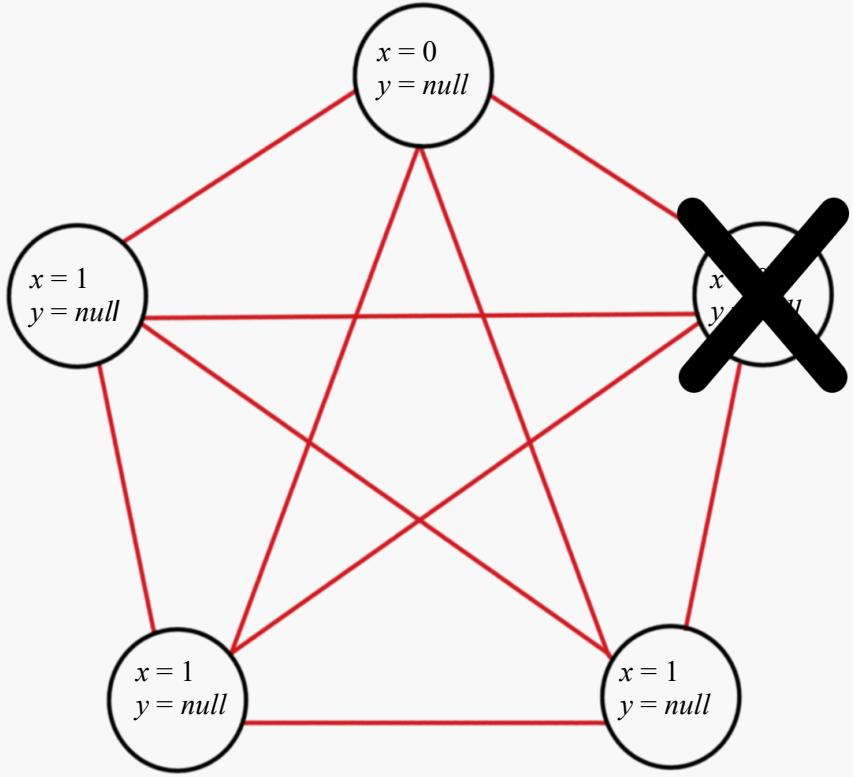
Esempio: fase 1, step 1

Gli n - *f* valori ricevuti non sono tutti uguali quindi ogni processo imposta ***y = null***



Esempio: fase 2, step 1

Ogni processo manda in **broadcast (*“second”, 1, y*)**

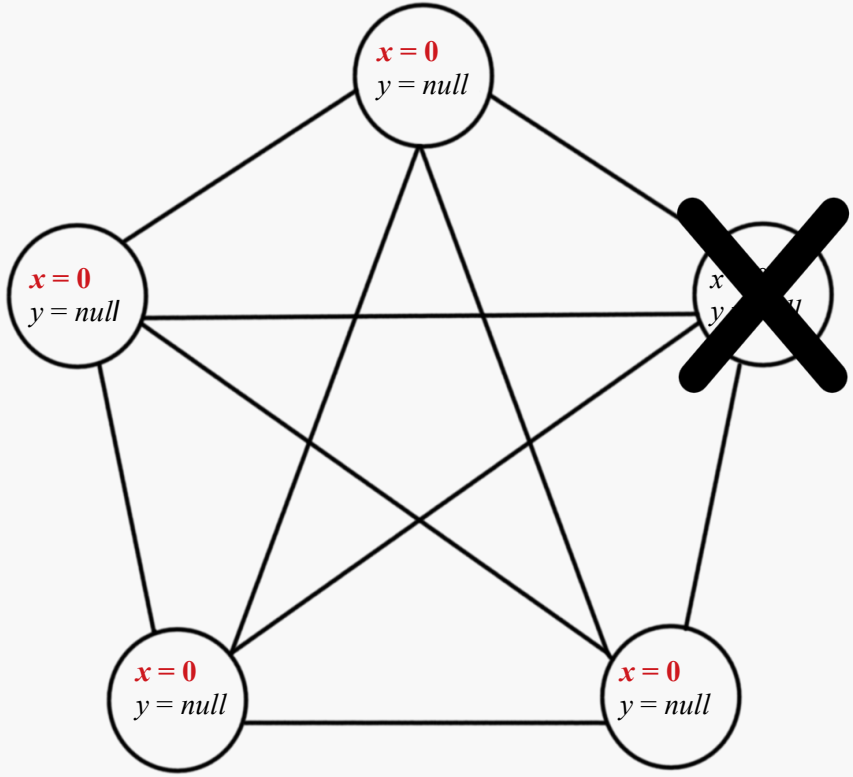


Esempio: fase 2, step 1

Tutti gli n - *f* messaggi ricevuti contengono *v* = *null*

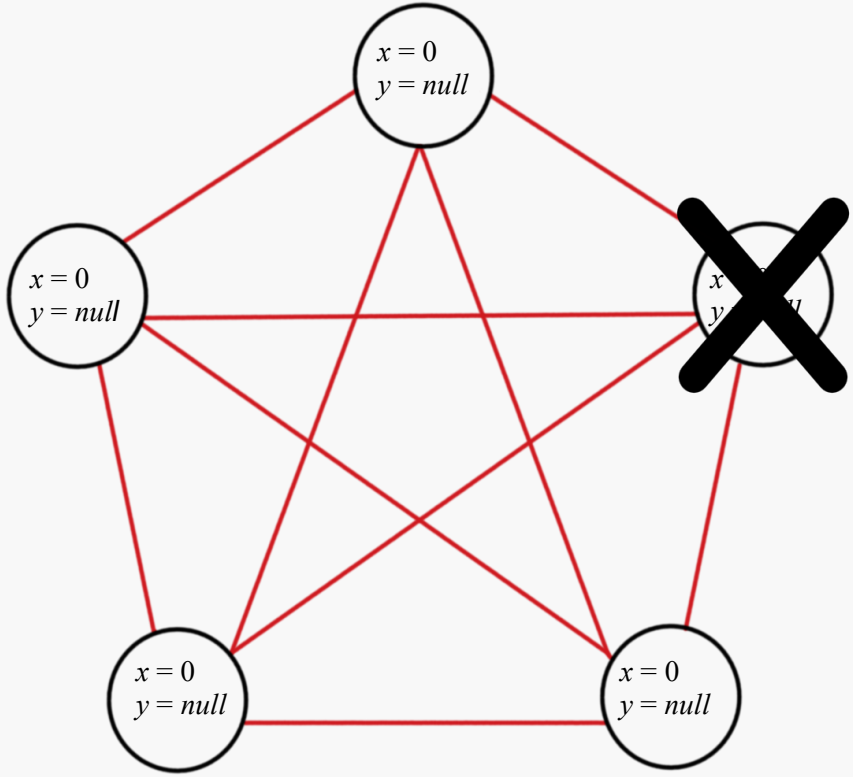
Quindi ogni processo **assegna ad *x* un valore casuale ∈ {0, 1}**

step = step + 1



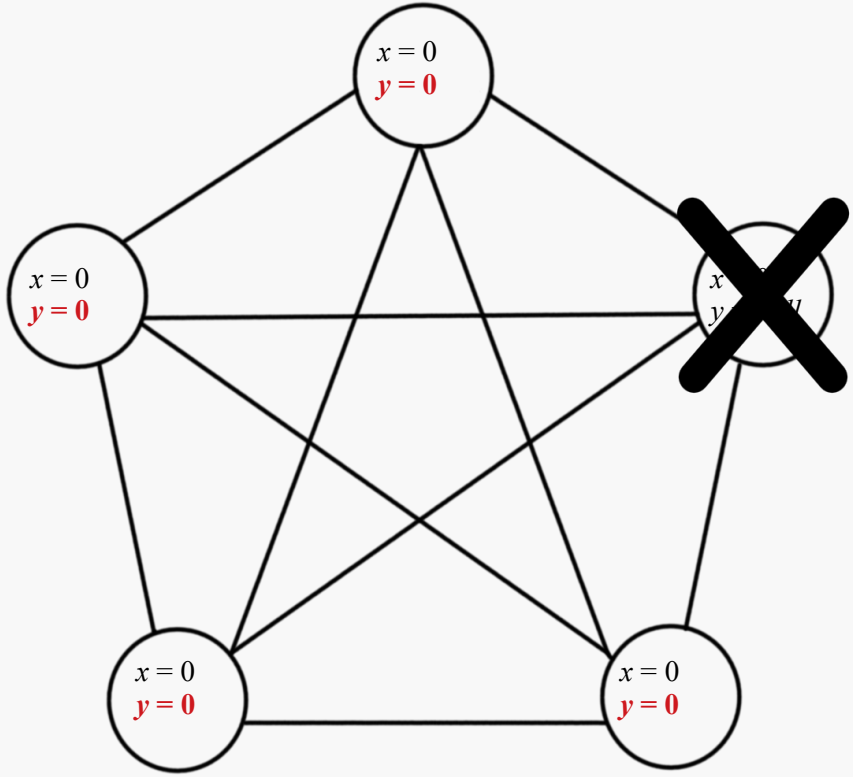
Esempio: fase 1, step 2

Ogni processo manda in **broadcast (*"first"*, *2*, *x*)**



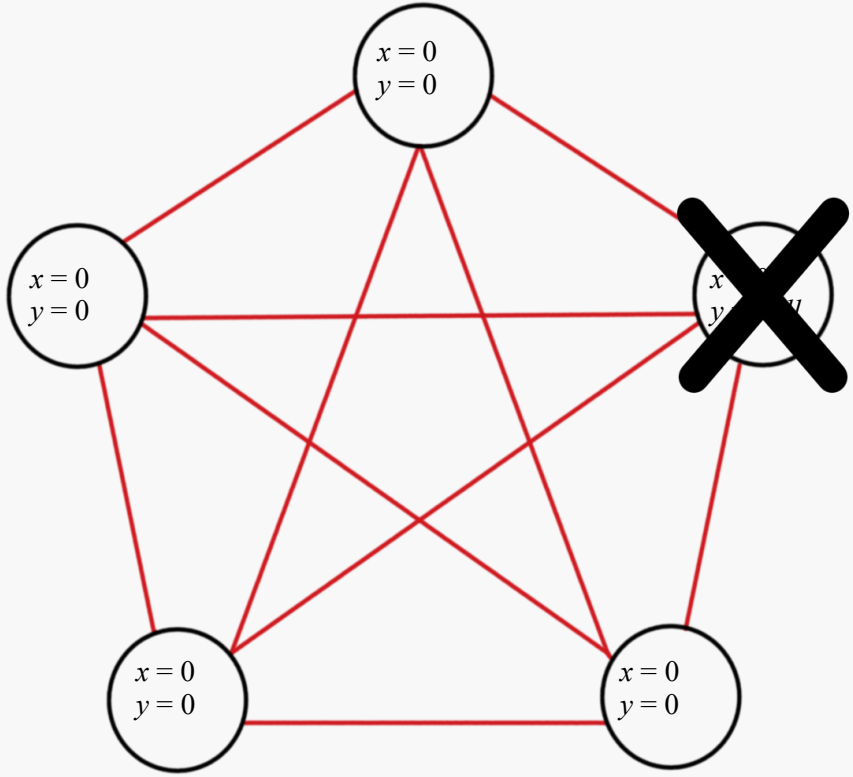
Esempio: fase 1, step 2

Tutti gli n - *f* messaggi ricevuti contengono lo stesso $v \neq null$ quindi ogni processo imposta $\boldsymbol{y = v}$ ($v = 0$)



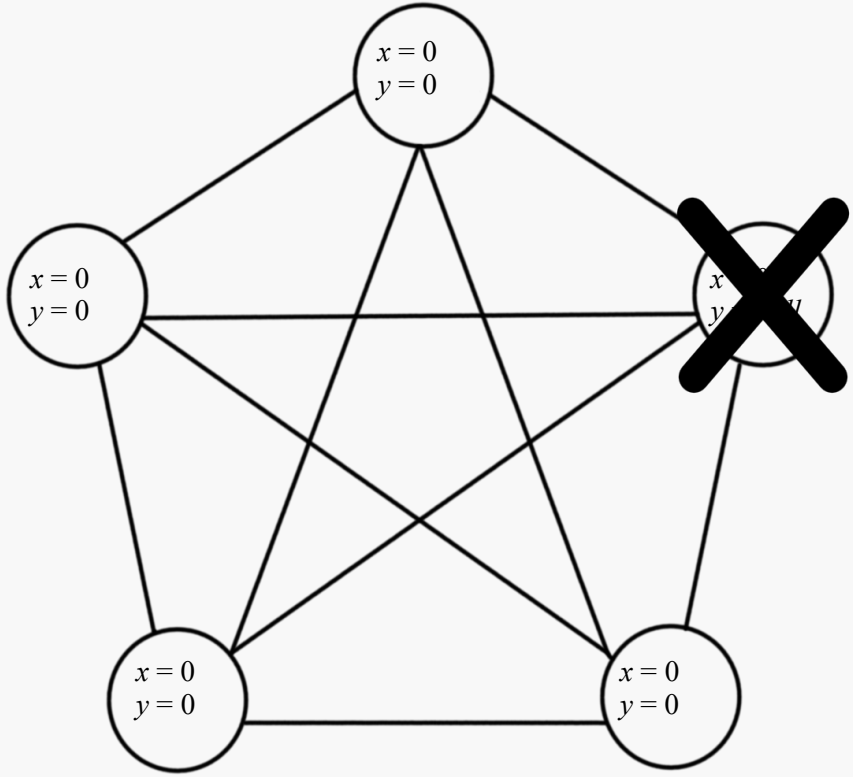
Esempio: fase 2, step 2

Ogni processo manda in **broadcast (*"second", 2, y*)**



Esempio: fase 2, step 2

Tutti gli n - *f* messaggi ricevuti contengono lo stesso $v \neq null$ quindi ogni processo esegue ***decide(v)*** *(*$v = 0$*)*



Correttezza dell'algoritmo

Well Formedness:

Ogni processo inizia con un'azione di $init(v)_i$ e quando eseguirà un'azione di *decide* non ripeterà piu nel seguito

Validity:

Se nella **fase 1** del primo round tutti i processi ricevono lo stesso valore di input v: allora nella **fase 2** tutti i processi compieranno l'azione di decide con lo stesso valore v

Agreement:

Si supponga che il processo P_i sia il primo a decidere e che questo avvenga nel round s; questo significa che ha ricevuto $n - f$ messaggi con lo stesso valore v. Tutti gli altri processi devono allora aver ricevuto almeno $n - 2f$ *messaggi* contenente il valore v, perciò essi non possono eseguire una decide per un valore diverso da v. In effetti essi possono soltanto:

- eseguire *decide(v)*
- impostare $x = v$ e compiere l'azione di *decide(v)* allo step successivo

Terminazione
È possibile un'esecuzione senza decisioni?

- Sia il numero di processi $n = 3f+1$ e sia m un numero compreso tra f e $2f$: $f < m \leq 2f$
- Si supponga che m processi partano con $x = 0$ e i restanti processi con $x = 1$
- Alla fine della fase 1 tutti i processi hanno $y = null$ e nella fase 2 tutti i processi scelgono il proprio valore di x casualmente
- Questa decisione casuale può portare ad una situazione in cui un numero m' di processi: $f < m' \leq 2f$ ha scelto $x = 0$ ed i restanti hanno $x=1$
- Si torna quindi alla situazione iniziale che può ripetersi all'infinito senza portare ad una decisione

Terminazione

Teorema

Per ogni $s > 0$, tutti i processi sani decideranno entro lo step $s+1$ con probabilità $\geq 1 - (1- ½^n)^s$

La prova procede per induzione sul valore di s.

- Per $s = 0$ ovvia ($p(t) \geq 0$).
- $s \geq 1$. Sia α la più corta esecuzione finita in cui un processo sano P_i ha ricevuto $n - f$ messaggi del tipo *("first", s, *)*.
  Se almeno $f + 1$ di questi messaggi contengono lo stesso valore v, v è detto valore "buono" per α.
  Ci possono essere uno o due valori buoni:
  - Un solo valore "buono". In questo caso ogni messaggio inviato nella fase due *("second", s, *)* conterra` lo stesso valore $v$ o *null*, quindi i processi che non impostano $x = v$ dovranno scegliere il valore casualmente. La probabilita` che i restanti processi scelgano lo stesso valore è $\geq ½^n$.
  - Due valori "buoni". In questo caso i processi sani avranno ricevuto sia il valore 0 che il valore 1 ed invieranno nella seconda fase il messaggio: *("second", s, null).* Di nuovo, la probabilità che tutti i processi scelgano in fase 2 lo stesso valore è $\geq ½^n$

Terminazione

Possiamo quindi affermare che la probabilità che tutti i processi sani scelgano, alla fine del round s, lo stesso valore e` $\geq \frac{1}{2^n}$. Quindi la probabilita` che non venga compiuta una scelta unica nello step s è:

$$\textbf{p(nessuna scelta unica)} \leq (1 - \tfrac{1}{2^n})$$

Poiché ogni step è indipendente, si ottiene che:

$$\textbf{p(nessuna scelta unica nei primi s step)} \leq (1 - \tfrac{1}{2^n})^s$$

Ne deriva che la probabilità che i processi sani decidano lo stesso valore *v* prima dello step *s*+1 è :

$$\textbf{p(unico valore deciso prima dello step s+1)} \geq 1 - (1 - \tfrac{1}{2^n})^s$$

Quindi la probabilità che tutti i processi sani compiano un'azione di *decide(v)* prima dello step *s*+1 è:

$$p(t) \geq 1 - (1 - \tfrac{1}{2^n})^s$$

Terminazione

$$p(t) \geq 1 - (1 - \tfrac{1}{2}^n)^s$$

per infiniti step:

$$s \rightarrow \infty: p = 1$$

L'algoritmo di Ben-Or garantisce la terminazione con probabilità $p(t) = 1$

Probabilità 1 ≠ Certezza

Se un evento ha probabilità 1 di verificarsi possiamo affermare che avverrà quasi sicuramente ma non ne abbiamo la certezza matematica

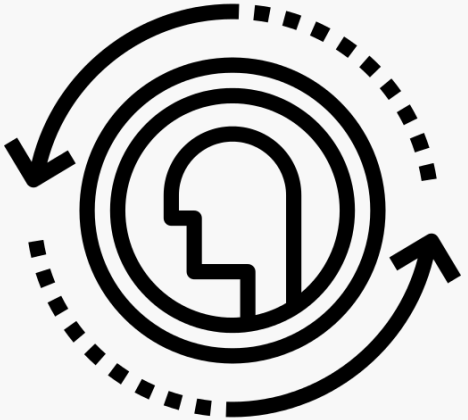
Esempio significativo:

Supponiamo di eseguire infiniti lanci di una moneta

La probabilità che esca solo croce equivale a 0 ($p = (½)^{\infty}$)

La probabilità che esca almeno una volta testa equivale a 1

Esiste comunque la possibilità di ottenere una sequenza composta da sole croci



Complessità della Terminazione

In ogni esecuzione fair in cui gli eventi di init avvengono su tutte le porte ogni processo che non fallisce esegue infiniti step

d è l'upper bound per il delivery time del broadcast del messaggio più vecchio

l è l'upper bound del tempo necessario per il completamento del task di ogni processo

Si può dire che ogni processo completa ogni volta un'iterazione *s* in $O(s(d+l))$ dall'ultimo evento di *init*

Global Coin Variant

Nell'algoritmo di Ben-Or un processo può scegliere allo step *s* un valore di *x* casuale (modellato come il lancio indipendente di una moneta {0; 1} per processo)

Se tutti i processi "estraggono" lo stesso valore, allora la decisione e la terminazione avviene all'iterazione successiva

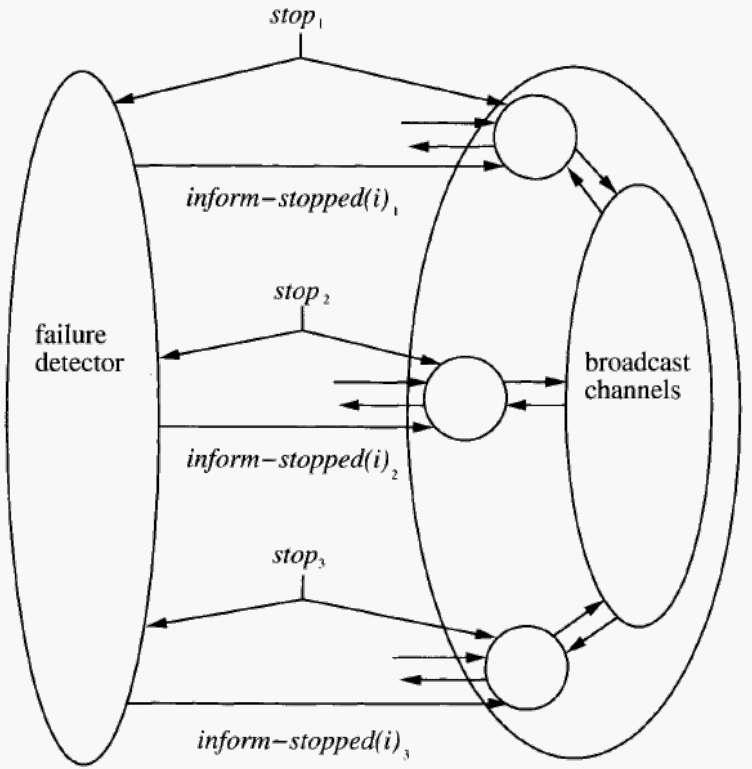
Possiamo immaginarci l'esistenza di una sola moneta, detta Global Coin, che tutti i processi lanciano allo stesso istante nel momento in cui uno o più processi deve assegnare un valore casuale ad *x*

Questa astrazione riduce drasticamente il tempo di esecuzione dell'algoritmo di Ben-Or

Si può dimostrare che nel caso di $n/3 \leq n < n/2$ e in presenza di un avversario "forte" (che può schedulare i processi) l'algoritmo ha probabilità di terminazione $p < 1$ (ma solo nei modelli con message passing!)

Failure Detector

- Metodo alternativo per la risoluzione del problema del consenso
- Modulo aggiunto in una rete asincrona che informa i processi sui fallimenti avvenuti
- Perfect FD: segnala tutti e solo i fallimenti avvenuti a tutti i processi attivi



Automa PerfectFDAgreement

Input: $init(v)_i$

$receive(w, I)_{j,i}$

$inform\text{-}stopped(j)_i$

Output: $bcast(w, I)_i$

$decide(\ v\ )_i$

Variabili:

$val \in W$, inizialmente *null*

$stopped \subseteq \{1,\dots, n\}$, inizialmente $\emptyset$

$ratified \subseteq \{1,\dots, n\}$, inizialmente $\emptyset$

decided, un Boolean, inizialmente *false*

Transizioni input

init(v)$_i$

Effetto: *val(i)* ← *v*

ratified ← {*i*}

inform-stopped(j)$_i$

Effetto: *stopped* ← *stopped* U {*j*}

ratified ← {*i*}

receive(w, I)$_{j,i}$

Effetto: **if** $j \notin$ *stopped* **then**

if (*w*, *I*) = (*val*, *stopped*) **then**

ratified ← *ratified* U {*j*}

else if (*w*, *I*) $>_d$ (*val*, *stopped*) **then**

stopped ← *stopped* U *I*

for all *k*, 1 ≤ *k* ≤ *n*, **do**

if *val(k)* = *null* **then** *val(k)* ← *w(k)*

ratified ← {*i*}

Transizioni output

$bcast(w, I)_i$

Precondizione: $w = val$

$I = stopped$

$val(i) \neq null$

Effetto: none

$decide(v)_i$

Precondizione: $ratified \ U \ stopped = \{1, \ldots, n\}$

$v = val(j)$

j = indice più piccolo con $val(j) \neq null$

$decided = false$

Effetto: $decided := true$

Proprietà dell'algoritmo

- **Well-formedness:** per costruzione
- **Validity:** evidente
- **Terminazione:** durante un'esecuzione *fair* (*init* su tutti i processi), dal momento che *val* e *stopped* possono essere modificati solo con l'aggiunta di informazione e che i processi eseguono un broadcast continuo dei propri valori, si giungerà ad un momento nel quale ognuno sarà in grado di effettuare la *decide*
- **Agreement**: In seguito alla prima *decide(v)* tutti i processi attivi condividono le stesse informazioni, affinché venga effettuata una *decide(w), $w \neq v$*, almeno un processo deve aver ricevuto dei valori differenti in input, ma questi potrebbero provenire solo da un processo *stopped* che però verrebbe ignorato

Grazie per l’attenzione

Borello Nazareno, Galatola Marco, Mecca Francesco