Impossibility of Consensus in Asynchronous Distributed systems where Processes can Crash

Frédéric TRONEL

IRISA - ADP Team

Friday 15th of June, 2001

Asynchronous Model

- A set of processes that communicate through reliable channels
- A single process can crash;
- Processes synchronize by exchanging messages;
- No global clock, neither shared memory;
- No bound neither on message transmission delays, nor on computation steps;

$\longrightarrow$ Simple and general model.

Informal Definition

- Agreement problem between processes;
- Processes are proposing an initial value that can be 0 or 1;
- Processes have to eventually agree on a common value, that must be chosen among the values initially proposed by them.

Formal Definition

Four properties have to be satisfied by any valid execution :

- Termination $\longrightarrow$ correct processes eventually decide;
- Integrity $\longrightarrow$ at most one decision by process;
- Validity $\longrightarrow$ the value decided must have been initially proposed;
- Agreement $\longrightarrow$ no two processes that decide, decide differently.

ε-agreement

Four properties have to be satisfied by any valid execution :

- Termination $\rightarrow$ correct processes eventually decide;
- Integrity $\rightarrow$ at most one decision by process;
- Validity $\rightarrow$ if all processes propose the same value, then this value must be eventually decided;
- Agreement $\rightarrow$ two processes are allowed to decide different value as long as their decision values are no more than ε apart from each other.

A simple protocol

- A simple protocol that solves ε-agreement despite one crash.

```
ε-agreement
begin
 (1)  r_i ← 0;
 (2)  est_i ← 0  or  1;
 (3)  while (r_i < log ε / log(n−1)) do
 (4)       send(proposition< p_i, r_i, est_i >) to all;
 (5)       wait until (n − 1) propositions have been received;
 (6)       est_i ← (Σ_j est_j) / (n−1);
 (7)       r_i ← r_i + 1;
 (8)  endo
 (9)  decide(est_i);
end
```

Why does it solve ε-agreement ?

Consider the distance between the estimation values of p_i and p_j after the k^{th} round :

$$d_k^{i,j} = \sqrt{(est_i^k - est_j^k)^2}$$

Why does it solve ε-agreement ?

We have :

$$
\begin{aligned}
\left(d_{k+1}^{i,j}\right)^2 &= (est_i^{k+1} - est_j^{k+1})^2 \\
&= \left(\sum_{l_i} \frac{est_{l_i}^k}{n-1} - \sum_{l_j} \frac{est_{l_j}^k}{n-1}\right)^2 \\
&= \frac{\left(\sum_{l_i} est_{l_i}^k - \sum_{l_j} est_{l_j}^k\right)^2}{(n-1)^2} \\
&\leq \frac{\left(est_{\sigma(i)}^k - est_{\sigma(j)}^k\right)^2}{(n-1)^2} \\
&\leq \frac{\left(d_k^{\sigma(i),\sigma(j)}\right)^2}{(n-1)^2}
\end{aligned}
$$

Why does it solve ε-agreement ?

- That means that the distance between estimations values is divided by at least $(n-1)$ each round. So this distance is less than ε (since initially it is less than 1) after :

$$\frac{\log \varepsilon}{\log(n-1)}$$

rounds.

- When all the processes propose the same value, this value is decided.

- We can solve ε-agreement.

Why cannot it solve consensus ?

- It would be very tempting to round the local decision value of each process back to 0 or 1.
- Since local decision values are very close, it might be possible that every process would end up with the same value.
- However this is not true, because of Sperner Lemma.

Why cannot it solve consensus ?

- Consider the situation around $\frac{1}{2}$. If p_i local decision value est_i is such that $\frac{1}{2} \leq est_i \leq \frac{1}{2} + \varepsilon$, it is possible that another process p_j has its local decision value est_j satisfying $\frac{1}{2} - \varepsilon < est_j \leq \frac{1}{2}$. Thus p_i is not allowed to round its value to 1 but to 0 (or conversely).

- However, in this case we have simply translate the problem from $\frac{1}{2}$ to $\frac{1}{2} + \varepsilon$.

- By induction, every process should have to decide 0.

- Even when all the initial values are 1. A contradiction.

Protocol Model

- Each process behavior is driven by an automaton.
- A process automaton can send (receive) messages.
- Each automaton has four distinguished states :
  - ⋆ Two initial states denoted I_0 et I_1.
  - ⋆ Two final states denoted F_0 et F_1.
- $F_{i \in \{0,1\}}$ are absorbing states.

Protocol Model

- An automaton can send a message (p_i, m) :
  - $\star$ $m \in M$ the set of possible messages.
- There exists a multiset denoted $\mathcal{M}$, that represents messages in transit, which is ruled by the two following operations :
  - $\star$ **send**(p_i, m) : $\mathcal{M} = \mathcal{M} \cup (p_i, m)$
  - $\star$ **receive**(p_i) : $\mathcal{M} = \mathcal{M} - a$ where a=$(p_i, m) \in \mathcal{M} \vee \emptyset$
- Calling **receive**(p_i) an infinite number of times leads to receive a message, if at least one message addressed to p_i is in transit.

Protocol Model

- A system *configuration* C is a couple $(s, \mathcal{M})$
  - $\star$ $s(i)$ is the state of automaton at process p_i.
  - $\star$ $\mathcal{M}$ multiset of messages in transit.

Protocol Model

- Let $C_1 = (s_1, \mathcal{M}_1)$, $C_2 = (s_2, \mathcal{M}_2)$ be two configurations

- A computation *step* $e = (p_i, m_1)$ can be applied to C_1 ($C_2 = e(C_1)$) iff :

  ⋆ p_i receive the message m_1 ($m_1 \in \mathcal{M}_1 \cup \emptyset$).

  ⋆ $\forall j \neq i \quad s_1(j) = s_2(j)$.

  ⋆ automaton at p_i moves from state $s_1(i)$ to state $s_2(i)$

  ⋆ p_i sends a message $m_2 \in M \cup \emptyset$.

  ⋆ $\mathcal{M}_2 = \mathcal{M}_1 - m_1 \cup m_2$.

- Note that m_2 is uniquely determined by $s_1(i)$ and m_1.

Protocol Model

- Let σ be a sequence of steps $(e_k)_{k\in I}$ (possibly infinite).

- σ can be applied to C and leads to $C' = C.\sigma$ iff :

$$\exists C_{k\in I} \quad \left\{ \begin{array}{rcl} C & = & C_0 \\ C_i & = & e_i(C_{i-1}) \\ C' & = & C_r \end{array} \right.$$

- $C' = \sigma(C)$ and C' is said to be *reachable* from C.

- A configuration C is said to be *accessible* iff it is reachable from an initial configuration.

Lemma 1

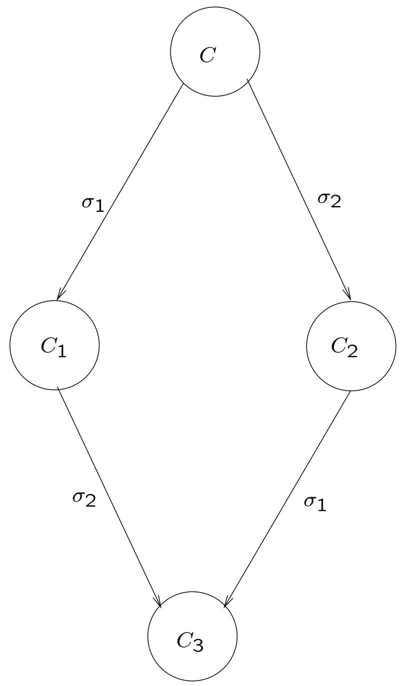
- Let C e a configuration.

- Let σ_1 (resp. σ_2) be a sequence that leads to C_1 (resp. C_2) from C.

- If the set of processes that take steps in both σ_1 et σ_2 are disjoint

- Then there exists a unique configuration C_3, such that σ_1 (resp. σ_2) can be applied to C_2 (resp. C_1) and leads to C_3. That is σ_1 and σ_2 commute.

Lemma 1



Protocol Model

- $C = (s, \mathcal{M})$ has a *decision value* $v \in \{0, 1\}$ if $\exists j$ such that $\quad s(j) = F_v$.

- A protocol P is partially correct iff :

  ⋆ No accessible configuration has more than one decision value.

  ⋆ $\forall v \in \{0, 1\}$, there exists an accessible configuration whose decision value is v.

Protocol Model

- Let σ be an infinite sequence of steps.
- A process p_i is *correct* in σ if it takes an infinite number of steps, otherwise it is *faulty*.
- σ is *admissible* iff :
  - ⋆ There is at most one faulty process in σ.
  - ⋆ All the messages sent to correct processes are received.

Protocol Model

- A protocol P is *correct* iff :
  - ⋆ It is partially correct.
  - ⋆ Every admissible execution of P has a single decision value.

The Result

There exists no correct consensus protocol.

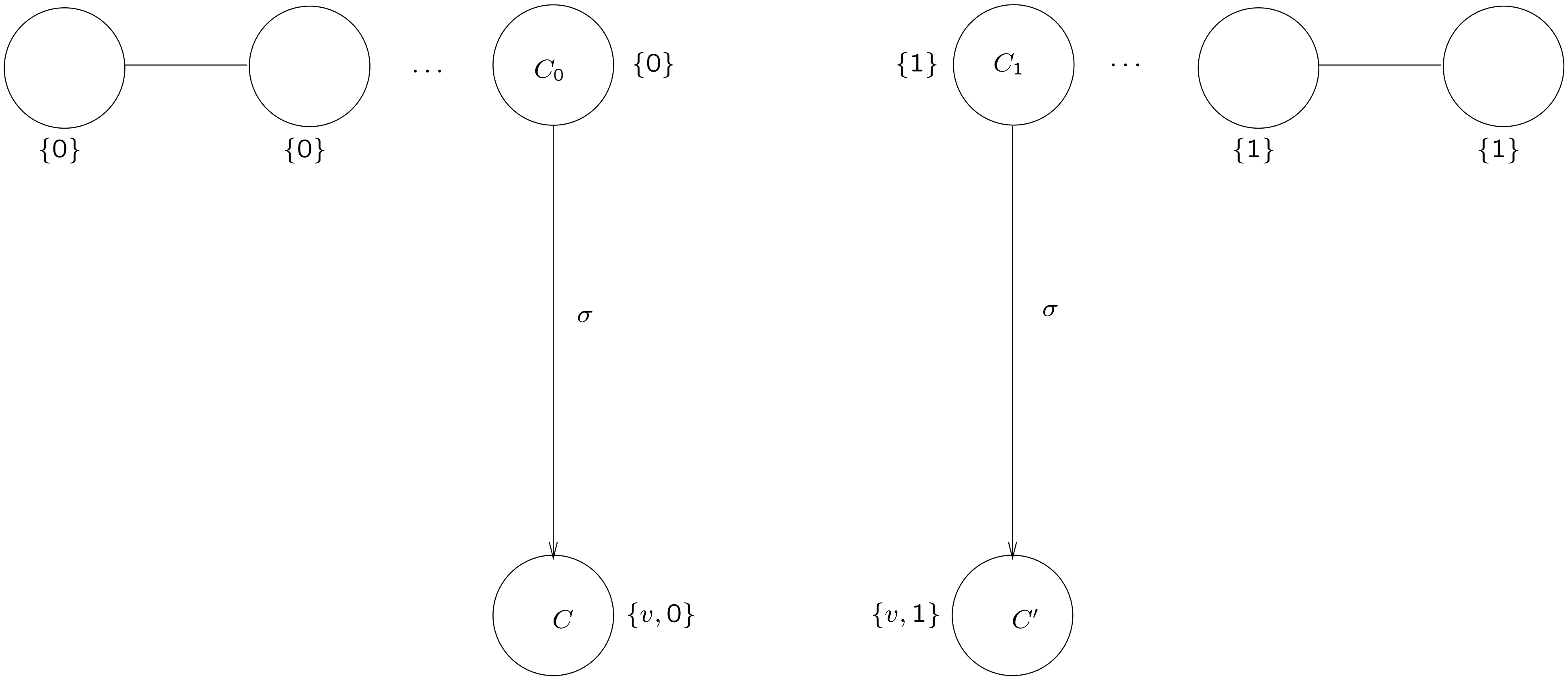
Notations

- Let $C = (s, \mathcal{M})$ be a configuration.
- Let V the set of decision values that can be obtained in any configuration reachable from C.
- C is bivalent if $\mid V \mid = 2$. Otherwise it is monovalent.
- If $V = v$, then C is v-valent.

Lemma 2

P has a bivalent initial configuration.

Proof



Lemma 3

- Let C be a bivalent configuration of P.
- Let $e = (p_i, m)$ be a step that can be applied to C.
- Let $\mathcal{C} = \{E \,|\, \exists \sigma$ such that $E = \sigma(C)$ and $p_i \notin \sigma\}$
- $\mathcal{D} = e(\mathcal{C})$
- Then $\mathcal{D}$ contains a bivalent configuration.

Proof of lemma 3

- Assume by contradiction that $\mathcal{D}$ only contains monovalent configurations.

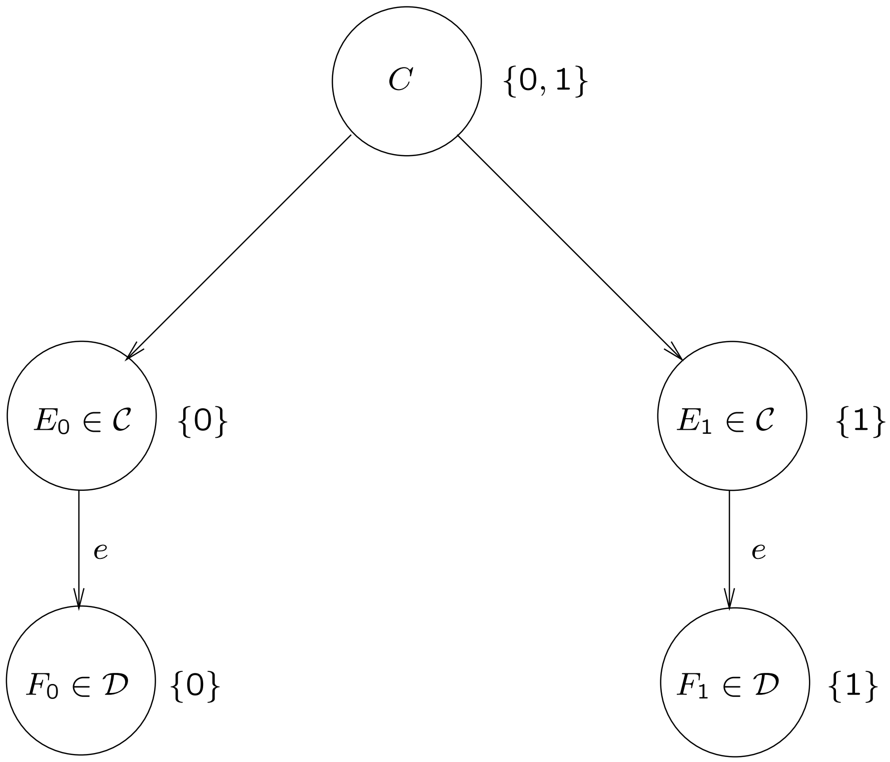
- Then $\mathcal{D}$ contains both 0-valent and 1-valent configurations :

  ⋆ By assumption C is bivalent, and so there exists two states E_i which are both reachable from C and i-valent.

  ⋆ We can consider two cases :

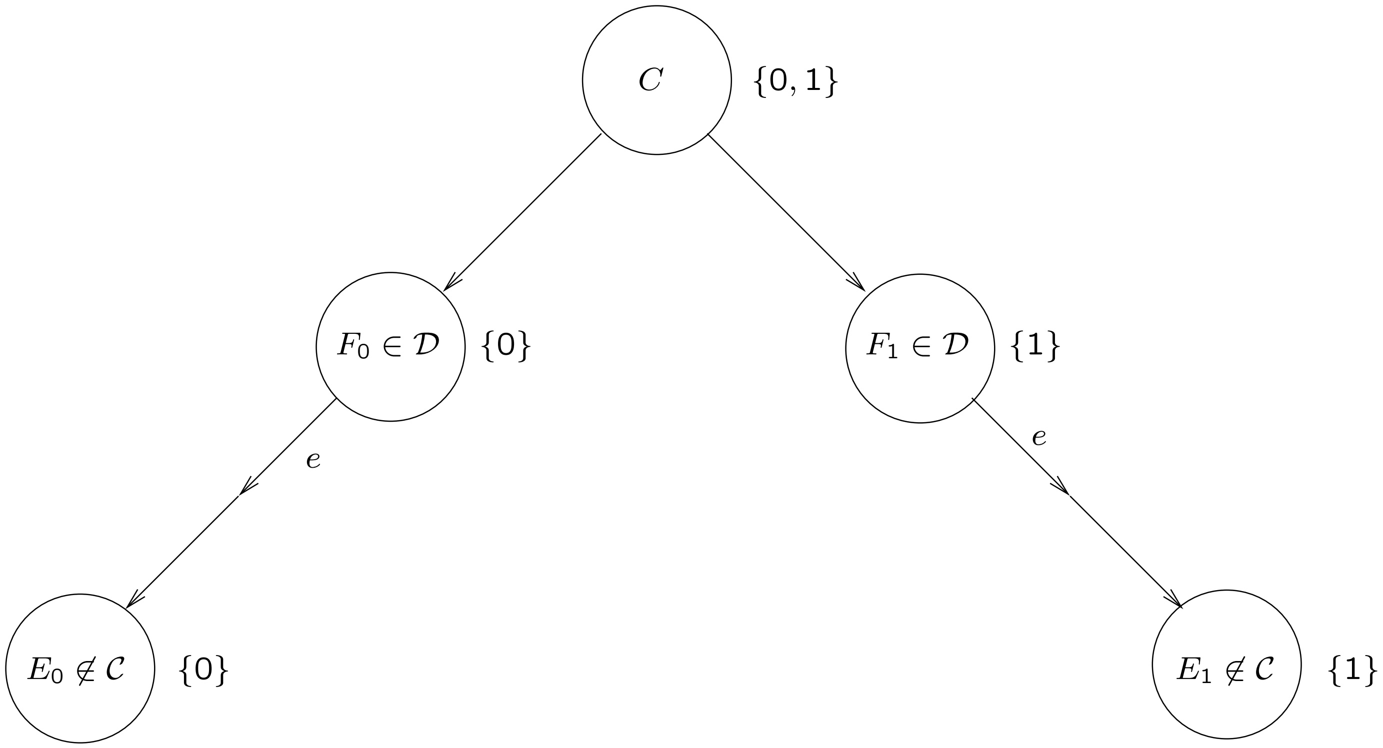
Proof of lemma 3

- E_i belongs to $\mathcal{C}$.



Proof of lemma 3

- E_i does not belongs to $\mathcal{C}$.
- That is, e was applied somewhere between C and E_i.



Proof of lemma 3

- Assume by contradiction that $\mathcal{D}$ only contains monovalent configurations.

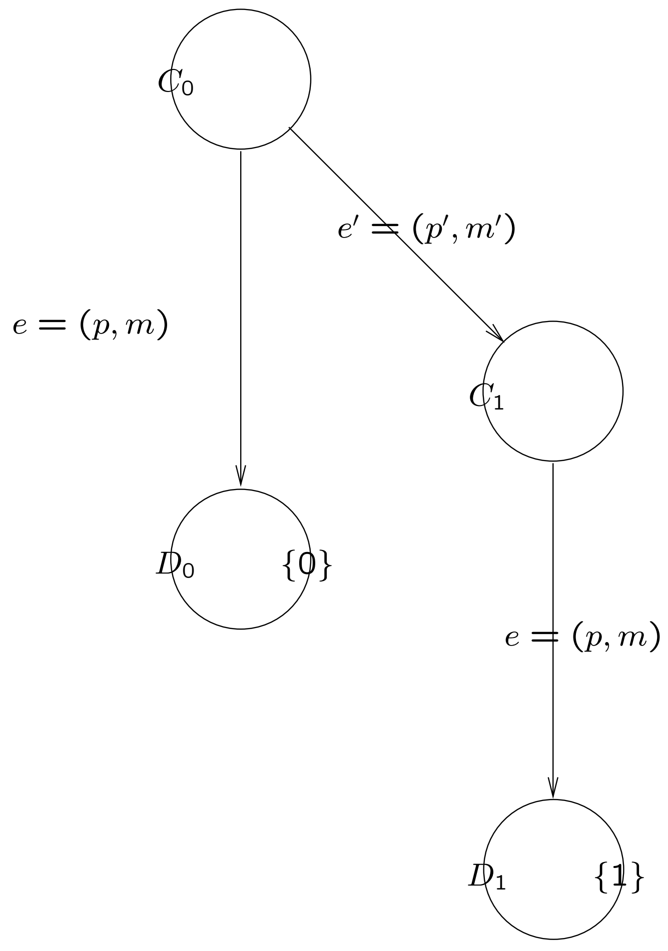
- Then $\mathcal{D}$ contains both 0-valent and 1-valent configurations.

- And there exists C_0 and C_1 in $\mathcal{C}$, such that :

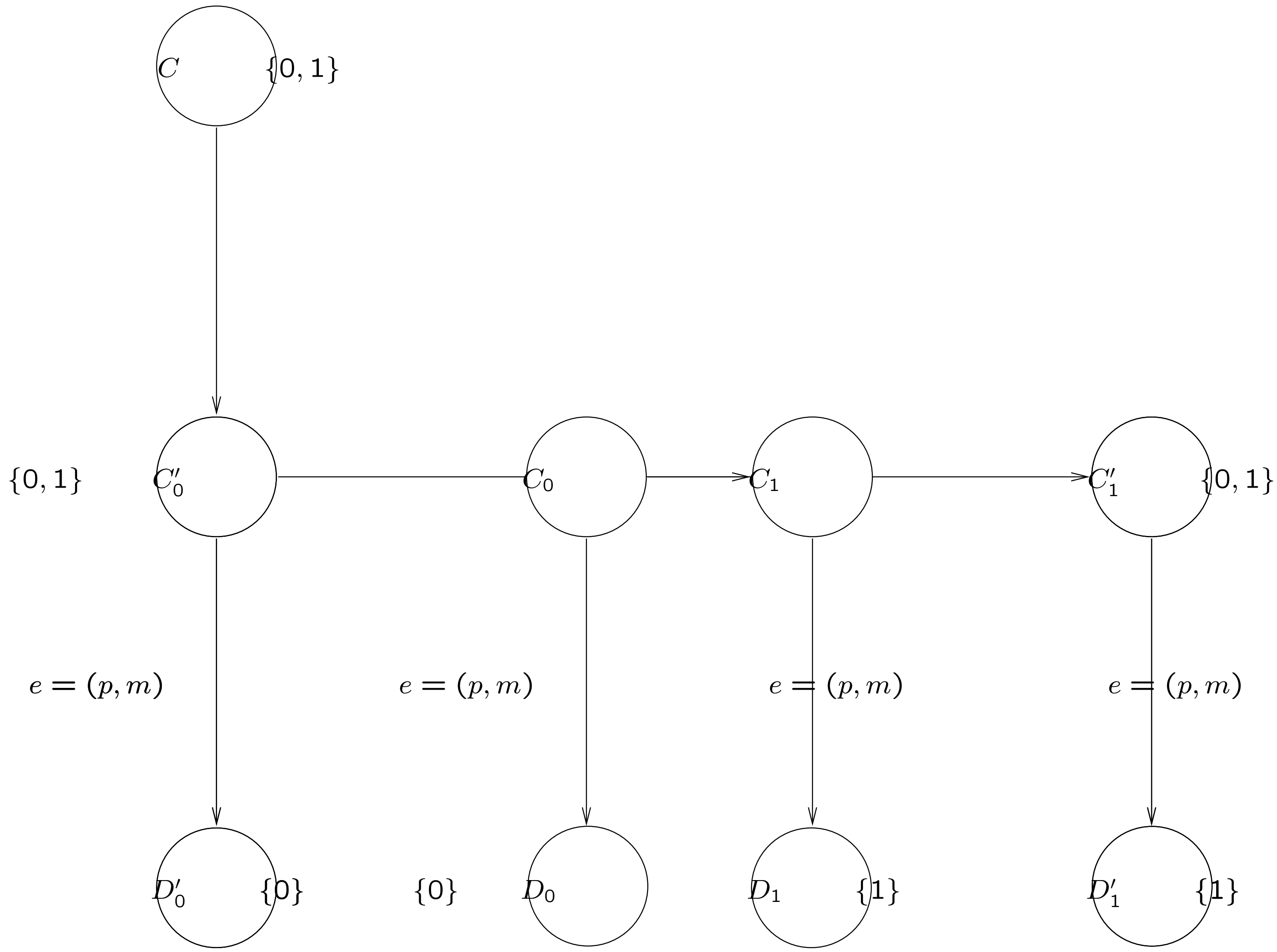
  ⋆ $D_i = e(C_i) \in \mathcal{D}$ is i-valent.

  ⋆ $C_1 = e'(C_0)$ with $e' = (p', m')$.

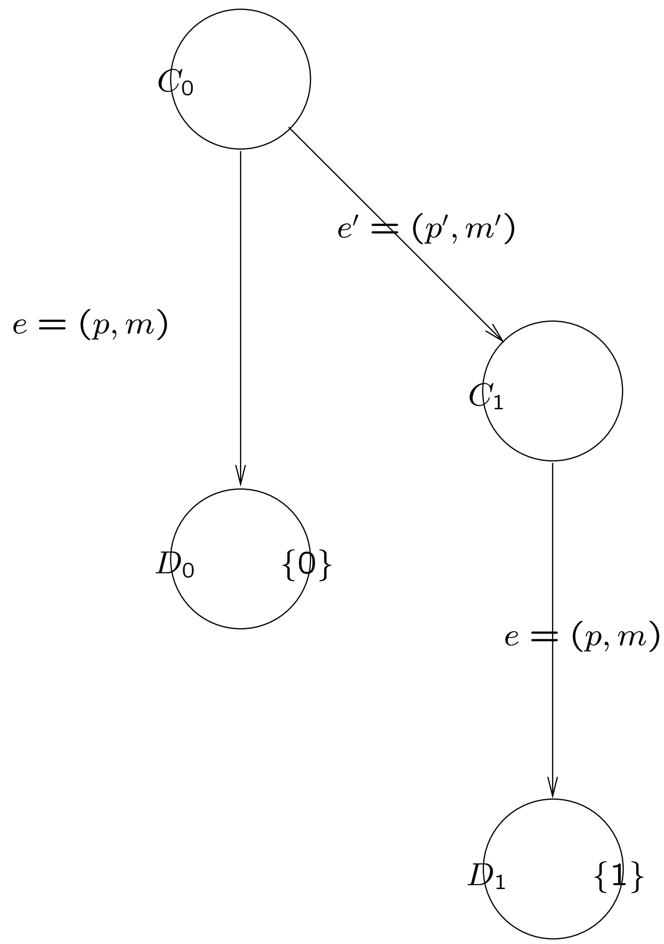
Proof of lemma 3



Proof of lemma 3



Proof of lemma 3

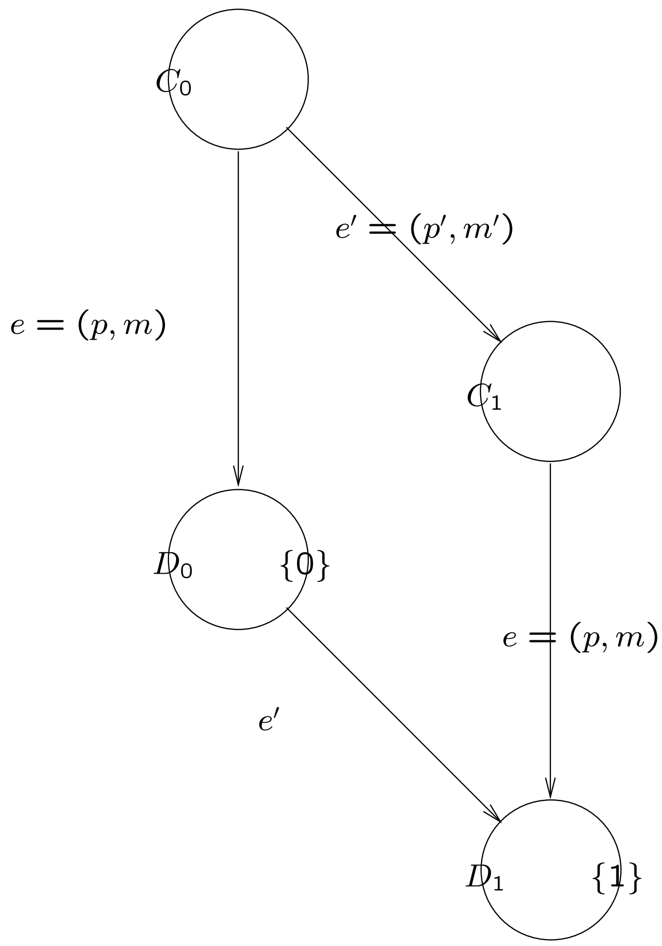


Proof of lemma 3

- Two cases have to be considered :
  - $\star$ either $p \neq p'$
  - $\star$ or $p = p'$.

Proof of lemma 3

- If $p \neq p'$ then $D_1 = e'(D_0)$ (Lemma 1), a contradiction.



Proof of lemma 3

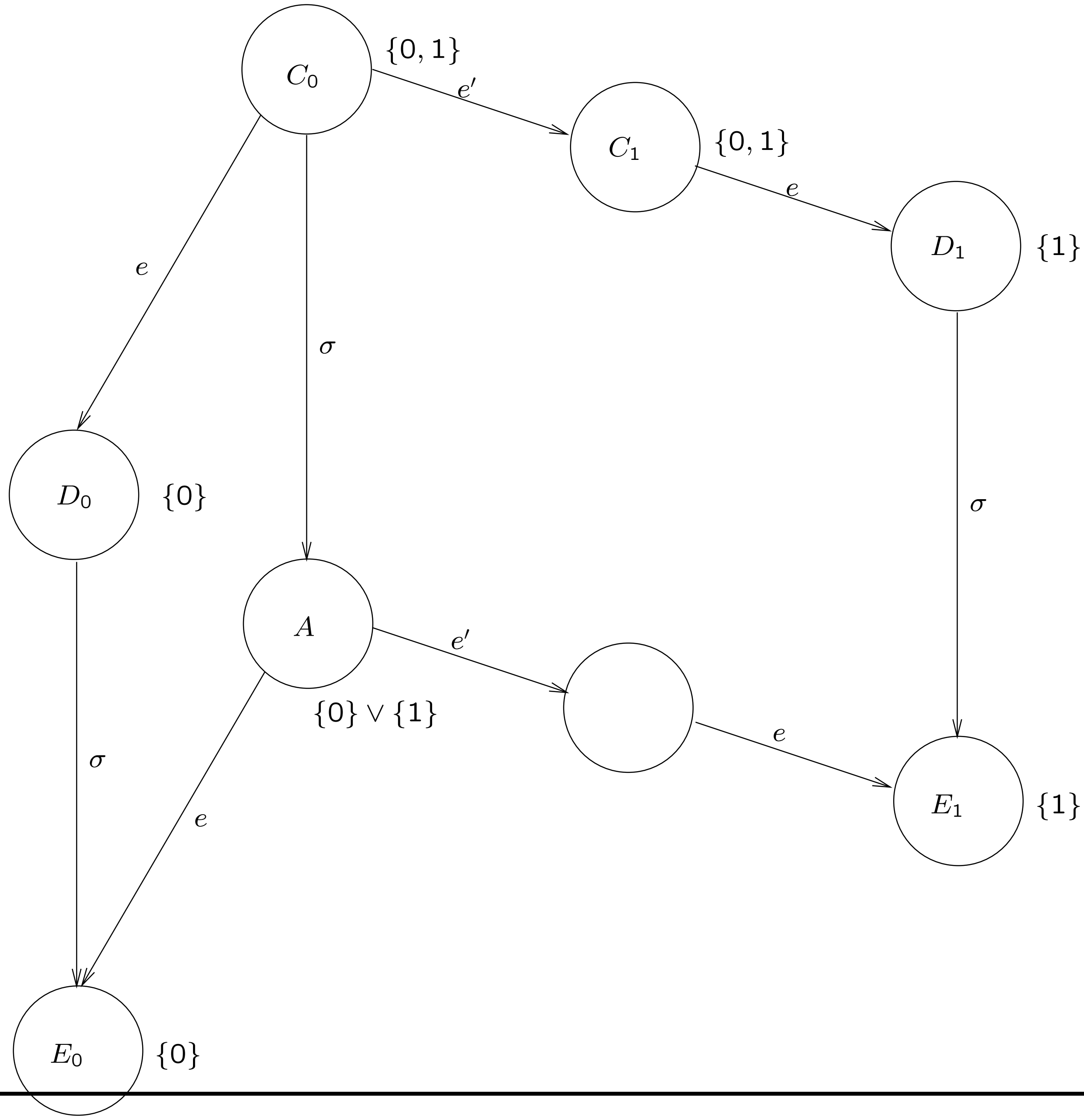
- So $p = p'$. Consider a finite sequence σ in which p does not take steps that can be directly applied to C_0. Let $A = \sigma(C_0)$.

  ⋆ σ can be applied to D_i and leads to $E_i = \sigma(D_i)$ (because p takes no steps in σ). Moreover, E_i is i-valent.

  ⋆ By lemma 1, $E_0 = e(A)$ and $E_1 = e(e'(A))$, so A must be bivalent, a contradiction.

- We conclude that $\mathcal{D}$ contains at least one bivalent configuration.

Proof of lemma 3



The end

- It is possible to build an admissible sequence which only has bivalent configurations. That is, no process decides.

- So, there exists no correct algorithm.

The end

- We choose a bivalent initial configuration $C_0 = (s_0, \emptyset)$ (Lemma 2). Let p be any process. By lemma 3, there exists a finite sequence σ :
  - $\star$ containing no steps of p (except the last one);
  - $\star$ starting from C_0;
  - $\star$ leading to a bivalent state $C = (s, \mathcal{M})$;
  - $\star$ whose last step is $(p, \emptyset)$.
- Reiterate the process with another process p chosen in a round-robin fashion, and with $m \in \mathcal{M}$ being the earliest message sent to p.