Iterated Models and Failure Detectors (Brief notes for discussion)

Alex Cornejo

CSAIL

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

Abstract

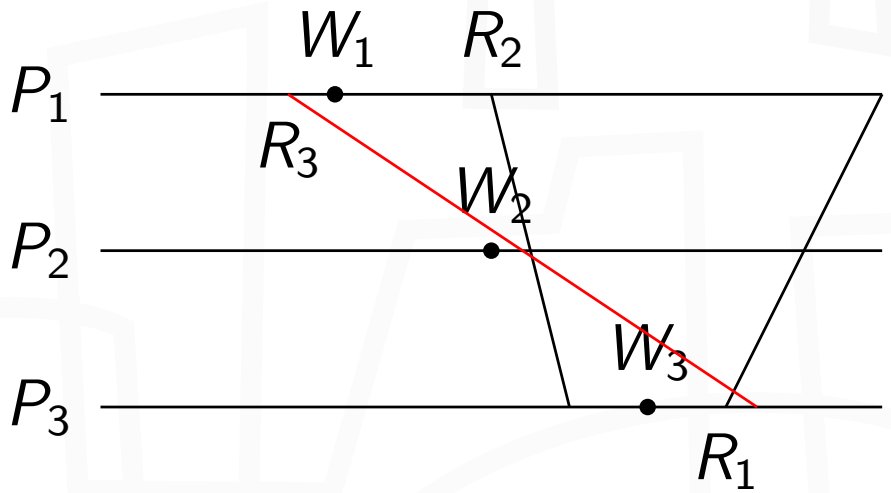
We analyze and extend the definition of failures detectors to iterated communication models. In particular we focus on iterated immediate snapshot model, but the results can be extended to iterated models in general. We start by examining the behavior of failures on iterated models and define the concepts of implicit failure and equivalence failure. We prove that without modification, the traditional definition of failure detectors does not augment the computational power of iterated models at the same rate that it does on non- iterated models. Finally we provide new definitions and algorithms to extent the concept of failures and failure detectors to iterated communication models.

Agenda

1. Communication models
   - Atomic snapshot
   - Immediate snapshot
   - Iterated models
2. Failures
   - Traditional Failures
   - Implicit failures
   - Equivalence failures
   - Failure detectors
3. IIS with failure detectors
   - Impossible to generalize simulation
   - IIS and equivalence failures
   - Future work

Atomic snapshot

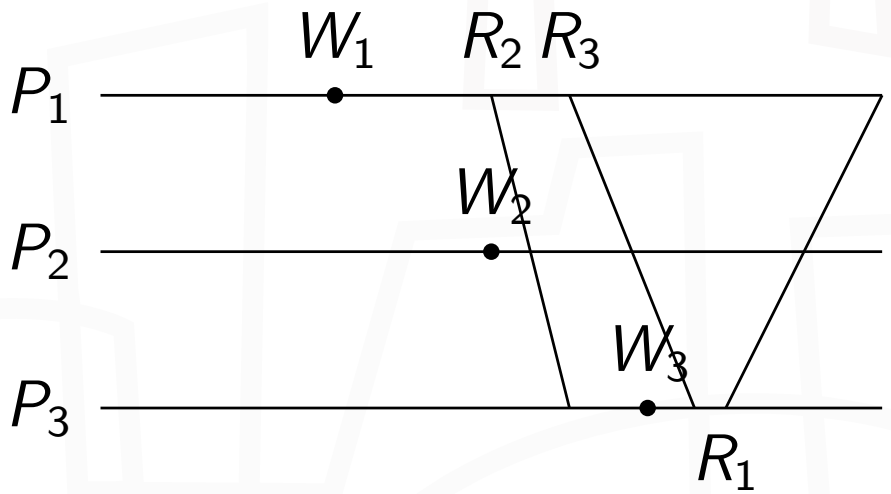
- In a system with n processes, an atomic snapshot object is built using n shared registers. This object provides two operations:
  1. Write: A process i writes a value v in register i.
  2. Read: A process i reads $1 \ldots n$ registers such that all vectors read are linearizable.



- It is possible to wrap this two operations in a single compound operation WriteRead, without loosing expressiveness.

Atomic snapshot

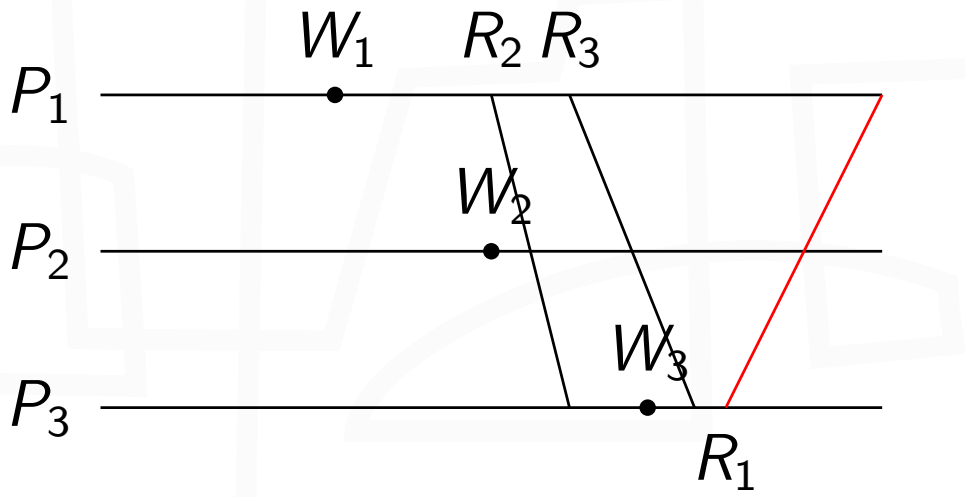
- In a system with n processes, an atomic snapshot object is built using n shared registers. This object provides two operations:
  1. Write: A process i writes a value v in register i.
  2. Read: A process i reads $1 \ldots n$ registers such that all vectors read are linearizable.



- It is possible to wrap this two operations in a single compound operation WriteRead, without loosing expressiveness.

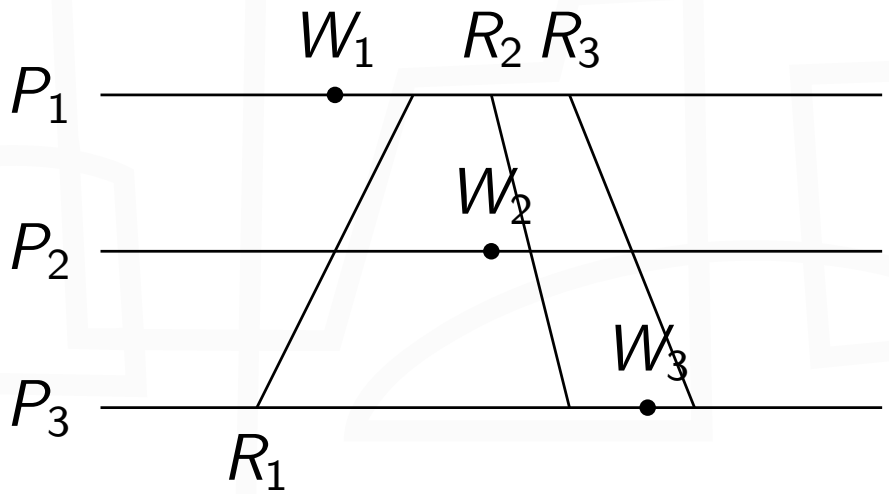
Immediate snapshot

- In a system with n processes, an immediate snapshot object is built using n shared registers. This object provides a single operation:
  1. Snapshot: Process i writes a value in register i and reads register $1 \ldots n$ such that the vector read is atomic and immediate with respect to the write.



Immediate snapshot

- In a system with n processes, an immediate snapshot object is built using n shared registers. This object provides a single operation:
  1. Snapshot: Process i writes a value in register i and reads register $1 \ldots n$ such that the vector read is atomic and immediate with respect to the write.



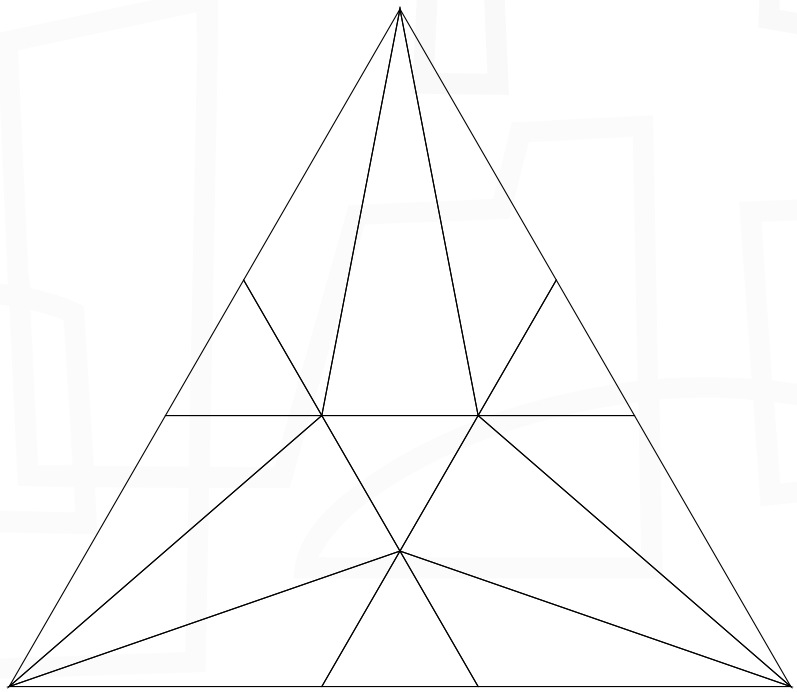
Iterated models

- Given a communication object of class M, an iterated model provides an array of instances $M_0, M_1, \ldots$, where each object is used consecutivly and each operation on M can only be used once (*one-shot*).
- In particular for the iterative immediate snapshot (IIS) model, the full information protocol looks like this:

1: $v \leftarrow \perp$, $w \leftarrow \perp$
2: **for** $i = 0, 1, \ldots$ **do**
3: $\quad w \leftarrow f(i, v, w)$
4: $\quad v \leftarrow M_i.\textsc{Snapshot}(w)$
5: **end for**

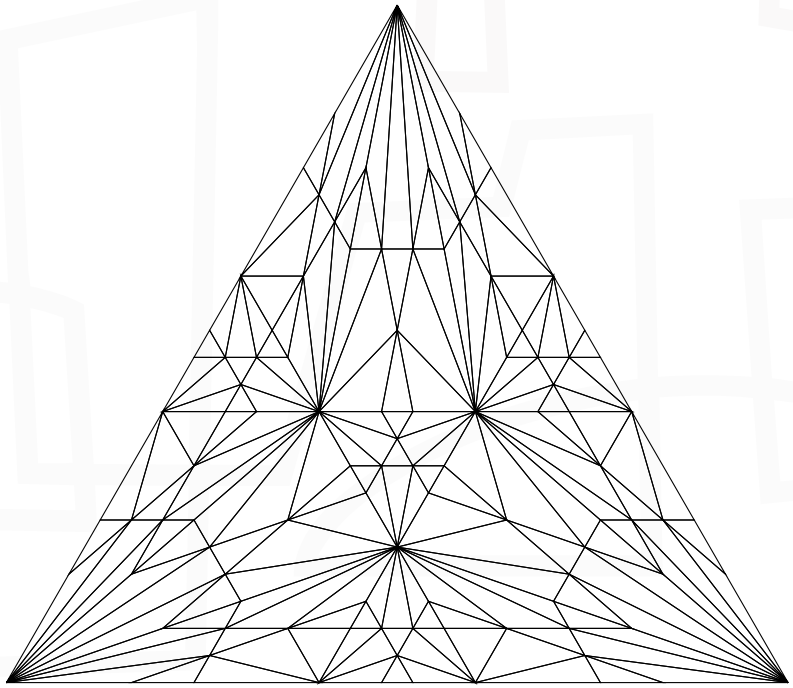
IIS properties

- In the iterative setting, the immediate snapshot properties can be restated more succintly.
  1. Self containment: $v_i \in S_i$
  2. Atomicity: $S_i \subseteq S_j$ or $S_j \subseteq S_i$
  3. $v_i \in S_j \rightarrow S_i \subseteq S_j$.



IIS properties

- In the iterative setting, the immediate snapshot properties can be restated more succintly.
  1. Self containment: $v_i \in S_i$
  2. Atomicity: $S_i \subseteq S_j$ or $S_j \subseteq S_i$
  3. $v_i \in S_j \rightarrow S_i \subseteq S_j$.



Traditional Failures

- We say that a process is faulty if its behaviour differs from the one described by the algorithm it is executing.
- Most research has focused on three types of failures:
  1. Crash: A crashed process stops its execution permaturly and never recovers.
  2. Omission: A process fails to receive or send a subset of messages.
  3. Byzantine: A byzantine process can behave arbitrarly (ie. conspire with other byzantine processes to sabotage execution)

Implicit Failures

- The notion of implicit failures can only be described within the context of a given model M and an existing failure definition DEF.
- Informally, implicit failures ocurr when there is an unrecoverable communication failure that escapes the definition off failure given by DEF in M.

Informal definition

A process $p \in \Pi$ in a given model M with a failure definition DEF suffers an implicit failure at time t if for all $t' > t$ it is unable to communicate with a subset $Q \subset \Pi$, and $\exists q \in Q$ such that q and p are correct under DEF.

Implicit Failures

Example

Supose a setting with two processes p and q in the IIS snapshot model where we only consider crash type failures, and both processes are correct. It is possible that $S_p \subsetneq S_q$ for all snapshots, and this is still a valid execution under this model and definition of failures. However, it is obvious that q is never seen by p, and thus there is an implicit failure.

Equivalence failures

- If under $\langle M, DEF \rangle$ there is room for implicit failures, this seems to suggest that somehow the definition of $\langle M, DEF \rangle$ is flawed.
- With equivalence failures we want to be able to define models $\langle M, EQ \rangle$ where implicit failures do not occurr.
- Informally, in an infinite execution, only the *smallest* set of processes thatexecute a protocol without deviating and exchange infinite information are considered correct by equivalence failures.

Definition

In an infinite execution let $P \subset \Pi$ be the set of processes that take an infinite number of steps without deviating from the algorithm. We can define a partial order over P where $p \preceq q$ if q recieves an infinite amount of information from p.

Failure detectors

- Failure detectors can be used as a bridge between syncrhonous and asyncrhonous systems.
- They are only an abstraction that encapsulates underlying assumptions about the system.
- Given a failure definition, we can define a failure pattern and a failure environment.
- Failure detectors are defined in the context of a given failure environment, and usually it is characterized in terms of completeness and accuracy properties.

Impossibility of general simulation

- There are algorithms to simulate *wait-free* atomic snapshot protocols in the iterative immediate snapshot model (IIS).
- However, it is trivial to prove that no such simulation exists for arbitrary protocols, therefore the iterative immediate snapshot model is strictly weaker than the atomic snapshot model. (explain)
- In a model augmented with failure detection, it is not reasonable to demand that processes continue to take steps independent of the progress of other processes.
- Given the two previous statements, it seems that the IIS model cannot be used with failure detectors.

IIS+$\mathcal{D}$ is weaker than IS+$\mathcal{D}$

- To prove this, it is sufficient to prove that there exists a task T which is solvable by the shared memory model augmented with $\mathcal{D}$ but that cannot be solved by IIS augmented with $\mathcal{D}$.

Informal proof.

Supose a protocol that solves consensus in an asyncrhonous system using an eventual leadership failure detector Ω. In IIS given Ω, supose that Ω behaves well from the begining and returns the id of a correct process ℓ. The executions where ℓ always executes the snapshot last and thus is never seen by any other process are all valid. If consensus were to be solved in these executions, this would imply that consensus can be solved without Ω.

IIS and equivalence failures

- The definition of equivalence failures solve part of the problem by eliminating implicit failures.
- However, even with equivalence failures the existing simulations fail to guarantee termination for correct processes.
- Within the definition of equivalence failures it is possible to construct an algorithm which guarantees termination for correct processes, even for non wait-free protocols.

Aided Converging Vector Algorithm

1: **function** AIDEDCONVERGINGVECTOR(v_i)
2:
3: $W_i \leftarrow v_i$
4:
5: **repeat**
6: $S_i \leftarrow M_{r_i}.Copia(W_i)$, $r_i \leftarrow r_i + 1$
7:
8:
9:
10:
11:
12:
13: **until** $f(S_i)$, $W_i \leftarrow S_i$
14: **return** W_i
15: **end function**

Aided Converging Vector Algorithm

1: **function** AIDEDCONVERGINGVECTOR(v_i)
2:
3: $W_i[i] \leftarrow v_i$
4: $F_i[i] \leftarrow F_i[i] + 1$
5: **repeat**
6: $S_i \leftarrow M_{r_i}.Copia(< F_i, W_i >)$, $r_i \leftarrow r_i + 1$
7:
8:
9:
10: **for** $j \in 1 \ldots n$ **do**
11: $k \leftarrow argmax\{F_k[j] | k \in \{1, \ldots, n\}\}$
12: $< F_i[j], W_i[j] > \leftarrow < F_k[j], W_k[j] >$
13: **until** $\forall < F, - > \in S_i,\ F = F_i$
14: **return** W_i
15: **end function**

Aided Converging Vector Algorithm

1: **function** AIDEDCONVERGINGVECTOR(v_i)
2: $W_i^P \leftarrow W_i, F_i^P \leftarrow F_i$
3: $W_i[i] \leftarrow v_i$
4: $F_i[i] \leftarrow F_i[i] + 1$
5: **repeat**
6: $S_i \leftarrow M_{r_i}.Copia(< F_i, W_i, F_i^P, W_i^P >)$, $r_i \leftarrow r_i + 1$
7: **if** $\exists F_j^P \in S_i.F_j^P[i] = F_i[i]$ **then**
8: $< F_i, W_i > \leftarrow < F_j^P, W_j^P >$
9: **return** W_i
10: **for** $j \in 1 \ldots n$ **do**
11: $k \leftarrow argmax\{F_k[j] | k \in \{1, \ldots, n\}\}$
12: $< F_i[j], W_i[j] > \leftarrow < F_k[j], W_k[j] >$
13: **until** $\forall < F, - > \in S_i,\ F = F_i$
14: **return** W_i
15: **end function**

Future work

- In the same vein as the notion of the hierarchy of failure detectors and communication objects, we can also define a hierarchy of failures.
- In this hierarchy, what is the weakest failure which allows the iterated models to simulate non iterated models.
- Given a failure detector inside the IIS, examine what changes from the topological perspective that allows the system to solve more problems.