Failure Detectors

Seif Haridi – Royal Institute of Technology
Peter Van Roy – Université catholique de Louvain

haridi(at)kth.se
peter.vanroy(at)uclouvain.be

1

Modeling Timing Assumptions

- It is tedious to model eventual synchrony (partial synchrony) over again in each algorithm
- Timing assumptions mostly needed to detect failures
  - Heartbeats, timeouts, etc…
- Use failure detectors to encapsulate timing assumptions
  - Black box giving suspicions regarding node failures
  - Accuracy of suspicions depends on model strength
  - Invention of failure detectors around 1990 was a major step toward the maturity of the field of distributed algorithms as opposed to a disorganized bunch of complicated algorithms

2

Implementation of Failure Detectors

- Typical Implementation
  - Periodically exchange heartbeat messages
  - Timeout based on worst case msg round trip
  - If timeout, then suspect node
  - If recv msg from suspected node, revise suspicion and increase time-out

Completeness and Accuracy

- Two important types of requirements
  - 1. Completeness requirements
    - Requirements regarding actually crashed nodes
      - When do they have to be detected?
  - 2. Accuracy requirements
    - Requirements regarding actually alive nodes
      - When are they allowed to be suspected?
- How to trivially achieve either? [d]
  - Both impossible in an asynchronous system!

Formal Model of FD

- Augment formal model with failure detectors (FD)
- A configuration consists of
  - State of each node
  - FD-state of each node
- Transition function on node i gets extra parameter:
  - FD-state of node i
- FD-state updated in comp(i) by another function
  - FD-function
  - Not modeled explicitly, but must satisfy some properties

Requirements: Completeness

- *Strong Completeness*
  - Every crashed node is *eventually* detected by *all* correct nodes
- There exists a time after which all crashed nodes are detected by all correct nodes
  - The book only studies detectors with this property
- Is it realistic? [d]

Requirements: Completeness

- *Weak Completeness*
  - Every crashed node is *eventually* detected by *some* correct node
- There exists a time after which all crashed nodes are detected by some correct node
  - A privileged node with good view of the others!
  - But: possibly detected by different correct nodes
    - Several privileged nodes, with good views to some

7

Requirements: Accuracy

- *Strong Accuracy*
  - No correct node is ever suspected
- For all nodes p and q,
  - p does not suspect q, unless q has crashed
- Is it realistic? [d]
  - Strong assumption, requires synchrony
  - I.e. no premature timeouts

8

Requirements: Accuracy

- *Weak Accuracy*
  - There exists a correct node which is never suspected by any node
- There exists a correct node P
  - Such that all nodes will never suspect P
- This is still quite a strong assumption
  - There is a privileged node that is always seen

Requirements: Accuracy

- *Eventual Strong Accuracy*
  - After some finite time the FD provides strong accuracy
- *Eventual Weak Accuracy*
  - After some finite time the detector provides weak accuracy
- After some time, the requirements are fulfilled
  - Prior to that, any behavior is possible!
- Quite weak assumptions [d]
  - When can eventual weak accuracy be achieved?

Four Main Established Detectors

- Four detectors with strong completeness
  - Perfect Detector (P)
    - Strong Accuracy
  - Strong Detector (S)
    - Weak Accuracy

  Synchronous Systems

  - Eventually Perfect Detector (◊P)
    - Eventual Strong Accuracy
  - Eventually Strong Detector (◊S)
    - Eventual Weak Accuracy

  Partially Synchronous Systems

11

Four Less Interesting Detectors

- Four detectors with weak completeness
  - Detector Q
    - Strong Accuracy
  - Weak Detector (W)
    - Weak Accuracy

  Synchronous Systems

  - Eventually Detector Q (◊Q)
    - Eventual Strong Accuracy
  - Eventually Weak Detector (◊W)
    - Eventual Weak Accuracy

  Partially Synchronous Systems

12

Interface of Perfect Failure Detector

- **Module:**
  - Name: PerfectFailureDetector (P)

- **Events:**
  - **Indication:** ⟨crash | p_i⟩
    - Notifies that node p_i has crashed

- **Properties:**
  - ***PFD1 (strong completeness)***
  - ***PFD2 (strong accuracy)***

Properties of P

- Properties:
  - ***PFD1 (strong completeness)***
    - Eventually every node that crashes is permanently detected by every correct node (liveness)
  - ***PFD2 (strong accuracy)***
    - If a node p is detected by any node, then p has crashed (safety)
- Safety or Liveness?

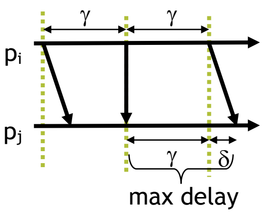
Implementing P in Synchrony

- Assume synchronous system
  - Max transmission delay between 0 and δ time units
- Each node every γ time units
  - Send <heartbeat> to all nodes
- Each node waits $\gamma+\delta$ time units
  - If did not get <heartbeat> from p_i
    - Detect <crash | p_i>

15

Correctness of P

- *PFD1 (strong completeness)*
  - A crashed node doesn't send <heartbeat>
    - Eventually every node will notice the absence of <heartbeat>
- *PFD2 (strong accuracy)*
  - Assuming local computation is negligible
  - Maximum time between 2 heartbeats
    - $\gamma+\delta$ time units
  - If alive, all nodes will recv hb in time
    - No inaccuracy



16

Interface of Eventually Perfect Failure Detector (◊P)

- **Module:**
  - Name: EventuallyPerfectFailureDetector (◊P)
- **Events:**
  - **Indication:** ⟨suspect | p_i⟩
    - Notifies that node p_i is suspected to have crashed
  - **Indication:** ⟨restore | p_i⟩
    - Notifies that node p_i is not suspected anymore
- **Properties:**
  - *PFD1 (strong completeness)*
  - *PFD2 (eventual strong accuracy)*. Eventually, no correct node is suspected by any correct node.

17

Implementing ◊P

- Assume partially synchronous system
  - Eventually some bounds exists
- Each node every γ time units
  - Send <heartbeat> to all nodes
- Each node waits T time units
  - If did not get <heartbeat> from p_i
    - Indicate <suspect | p_i> if p_i is not in **suspected**
    - Put p_i in **suspected** set
  - If get HB from p_i, and p_i is in **suspected**
    - Indicate <restore | p_i> and remove p_i from **suspected**
    - Increase timeout **T**

18

Correctness of ◊P

- *EPFD1 (strong completeness)*
  - Same as before
- *EPFD2 (eventual strong accuracy)*
  - Each time p is inaccurately suspected by a correct q
    - Timeout T is increased at q
    - Eventually system becomes synchronous, and T becomes larger than the unknown bound δ ($T>\gamma+\delta$)
    - q will receive HB on time, and never suspect p again

Leader Election

… example of "specification engineering"

Leader Election vs Failure Detection

- Failure detection captures failure behavior
  - Detect failed nodes
- Leader election (LE) also captures failure behavior
  - Detect correct nodes (a single & same for all)
- Formally, leader election is a FD
  - Always suspects all nodes except one (leader)
  - Ensures some properties regarding that node

Leader Election vs Failure Detection

- We'll define two leader election algorithms
  - Leader election (LE) which "matches" P
  - Eventual leader election (Ω) which "matches" ◊P

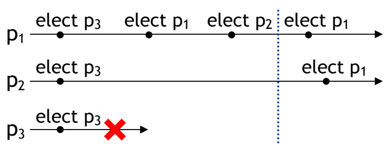
Matching LE and P

- P's properties
  - P always eventually detects failures (strong completeness)
  - P never suspects correct nodes (strong accuracy)
- Completeness of LE
  - Informally: eventually ditch crashed leaders
  - Formally: eventually every correct node trusts some correct node
- Accuracy of LE
  - Informally: never ditch a correct leader
  - Formally: No two correct nodes trust different correct nodes
    - Is this really accuracy? [d]
    - Yes! Assume two nodes trust different correct nodes
      - One of them must eventually switch, i.e. leaving a correct node

23

LE desirable properties

- LE always eventually detects failures
  - Eventually every correct node trusts some correct node
- LE is always accurate
  - No two correct nodes trust different correct nodes
- But the above two permit the following

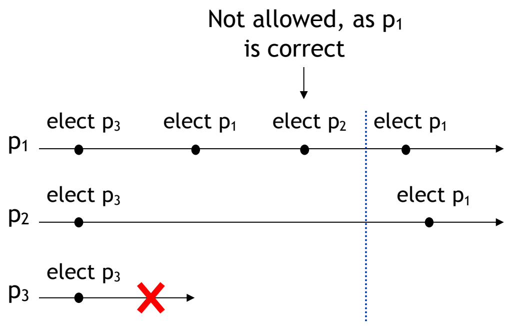


- But P_1 is "inaccurately" leaving a correct leader

24

LE desirable properties

- To avoid "inaccuracy" we add
  - Local Accuracy:
    - If a node is elected leader by p_i, all previously elected leaders by p_i have crashed



25

Interface of Leader Election

- **Module:**
  - Name: LeaderElection (le)
- **Events:**
  - **Indication:** ⟨leLeader | p_i⟩
    - Indicate that leader is node p_i
- **Properties:**
  - ***LE1 (eventual completeness).*** Eventually every correct node trusts some correct node
  - ***LE2 (agreement).*** No two correct nodes trust different correct nodes
  - ***LE3 (local accuracy).*** If a node is elected leader by p_i, all previously elected leaders by p_i have crashed

26

Implementing LE

- Globally rank all nodes
  - E.g. rank ordering $p_1 > p_2 > p_3 > p_4$
  - Represented by function r which returns all nodes with higher ranking
  - $r(p_1) = \emptyset$,
  - $r(p_2) = \{p_1\}$,
  - $r(p_3) = \{p_1, p_2\}$
  - $r(p_4) = \{p_1, p_2, p_3\}$

Implementing LE (2)

- **Implements:** LeaderElection (le).
- **Uses:** PerfectFailureDetector (P).

- **upon event** ⟨init⟩ **do**
  - suspected = ∅; leader := highest(r)
  - **trigger** ⟨leLeader | leader⟩

- **upon event** ⟨crash | p_i⟩ **do**
  - suspected := suspected ∪ $\{p_i\}$

- **upon exists** p_i **s.t.** $r(p_i) \subseteq$ suspected $\wedge$ $p_i \notin$ suspected **do**
  - leader := p_i
  - **trigger** ⟨leLeader | p_i⟩

Matching Ω and ◊P

- ◊P weakens P by only providing eventual accuracy
  - Weaken LE to Ω by only guaranteeing eventual agreement

eventual

- **LE Properties:**
  - ***LE1 (eventual completeness).*** Eventually every correct node trusts some correct node
  - ***LE2 (agreement).*** No two correct nodes trust different correct nodes
  - ~~***LE3 (local accuracy).*** If a node is elected leader by p_i, all previously elected leaders by p_i have crashed~~

29

Interface of Eventual Leader Election

- **Module:**
  - Name: EventualLeaderElection (Ω)
- **Events:**
  - **Indication**: ⟨leLeader | p_i⟩
    - Notify that p_i is trusted to be leader
- **Properties:**
  - ***ELD1 (eventual completeness).*** Eventually every correct node trusts some correct node
  - ***ELD2 (eventual agreement).*** Eventually no two correct nodes trust different correct nodes

30

Eventual Leader Detection Ω

- In crash-stop process abstraction
  - Ω is obtained directly from ◊P
  - Each node trusts the node with highest id among all nodes not suspected by ◊P
  - Eventually, exactly one correct process will be trusted by all correct processes

Implementing Ω

- **Implements:** EventualLeaderElection (Ω).
- **Uses:** EventuallyPerfectFailureDetector (◊P).

- **upon event** ⟨init⟩ **do**
  - suspected := ∅; leader := p_n;
  - **trigger** ⟨leLeader | leader⟩

- **upon event** ⟨suspect | p_i⟩ **do**
  - suspected := suspected ∪ {p_i}

- **upon event** ⟨restore | p_i⟩ **do**
  - suspected := suspected \ {p_i}

- **upon exists** p_i **s.t.** $r(p_i) \subseteq$ suspected $\wedge$ $p_i \not\in$ suspected **do**
  - leader := p_i
  - **trigger** ⟨leLeader | p_i⟩

Ω for Crash Recovery

- Can we elect a recovered node? [d]
  - Not if it keeps crash-recovering infinitely often!
- Basic idea
  - Count number of times you've crashed (**epoch**)
  - Distribute your **epoch** periodically to all nodes
  - Elect leader with lowest **(epoch, node_id)**
- Implementation
  - Similar to ◊P and Ω for crash-stop
  - Piggyback **epoch** with heartbeats
  - Store and load leader upon crash

33

Reductions

34

Reductions

- We say $X \preccurlyeq Y$ if
  - X can be solved given a solution of Y
  - Read X is reducible to Y
  - Informally, problem X is easier than or as hard as Y

Preorders, partial orders…

- A relation ~ is a preorder on a set A if for any x,y,z in A
  - x~x (reflexivity)
  - x~y and y~z implies x~z (transitivity)

- Difference between preorder and partial order
  - Partial order is a preorder with antisymmetry
    - x~y and y~x implies x=y
  - I.e. two different objects x and y cannot be symmetric
    - i.e. it isn't possible that x~y and y~x for two different x and y

$\preccurlyeq$ is a preorder

- $\preccurlyeq$ is a preorder
  - Reflexivity. $X \preccurlyeq X$
    - X can be solved given a solution to X
  - Transitivity. $X \preccurlyeq Y$ and $Y \preccurlyeq Z$ implies $X \preccurlyeq Z$
    - Since $Y \preccurlyeq Z$, use impl. of Z to impl. Y.
      use impl. of Y to impl. X. Hence we impl. X from Z's impl.
- $\preccurlyeq$ is not antisymmetric, thus not a partial order
  - Two different X and Y can be equivalent
    - Distinct problems X and Y can be solved from the other's solution

37

Shortcut definitions

- We write $X \simeq Y$ if
  - $X \preccurlyeq Y$ and $Y \preccurlyeq X$
  - Problem X is equivalent to Y
- We write $X \prec Y$ if
  - $X \preccurlyeq Y$ and not $X \simeq Y$
  - or equivalently, $X \preccurlyeq Y$ and not $Y \preccurlyeq X$
  - Problem X is strictly weaker than Y, or
  - Problem Y is strictly stronger than X

38

Example

- It is true that ◊P≼P
  - Given P, we can implement ◊P
    - We just return P's suspicions.
    - P always satisfies ◊P's properties
- In fact, ◊P≺P in the asynchronous model
  - Because it is not true that P≼◊P
- Reductions common in computability theory
  - If X≼Y, and if we know X is impossible to solve
    - Then Y is impossible to solve too
  - If ◊P≼P, and some problem Z can be solved with ◊P
    - Then Z can also be solved with P

Weakest FD for a problem?

- Often P is used to solve problem X
  - But P is not very practical (needs synchrony)
  - Is X a "practically" solvable problem?
    - Can we implement X with ◊P?
    - Sometimes a weaker FD than P will not solve X
      - Proven using reductions

Weakest FD for a problem?

- We might know that $X \preccurlyeq P$ (X solvable with P)
  - Can we solve X with a weaker FD than P, say $\Diamond P$?
  - Or is it impossible, i.e. $\Diamond P \prec X$
- Common proof to show P is weakest FD for X
  - Prove that $P \preccurlyeq X$
  - I.e. P can be solved given X
- If $P \preccurlyeq X$ then $\Diamond P \prec X$
  - Because we know $\Diamond P \prec P$ and $P \simeq X$, i.e. $\Diamond P \prec P \simeq X$
    - If we can solve X with $\Diamond P$, then
    - we can solve P with $\Diamond P$, which is a contradiction

How are the detectors related?

Requirements: Completeness

- *Strong Completeness*
  - Every crashed node is *eventually* detected by *all* correct nodes
- There exists a time after which all crashed nodes are detected by all correct nodes
  - The book only studies detectors with this property
- Is it realistic? [d]

43

Requirements: Completeness

- *Weak Completeness*
  - Every crashed node is *eventually* detected by *some* correct node
- There exists a time after which all crashed nodes are detected by some correct node
  - A privileged node with good view of the others!
  - But: possibly detected by different correct nodes
    - Several privileged nodes, with good views to some

44

Requirements: Accuracy

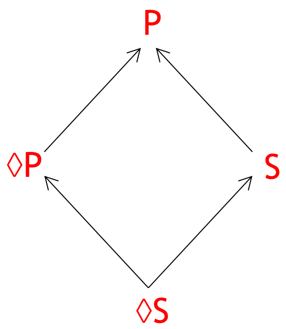
- *Strong Accuracy*
  - No correct node is ever suspected
- For all nodes p and q,
  - p does not suspect q, unless q has crashed
- Is it realistic? [d]
  - Strong assumption, requires synchrony
  - I.e. no premature timeouts

Requirements: Accuracy

- *Weak Accuracy*
  - There exists a correct node which is never suspected by any node
- There exists a correct node P
  - Such that all nodes will never suspect P
- This is still quite a strong assumption
  - There is a privileged node that is always seen

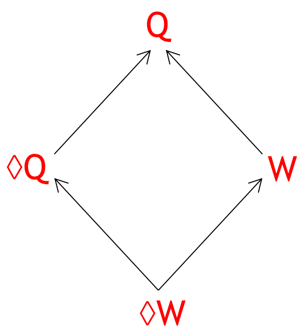
Trivial Reductions

- Strongly complete
  - $\Diamond P \preccurlyeq P$
    - P is always strongly accurate, thus also eventually strongly accurate
  - $\Diamond S \preccurlyeq S$
    - S is always weakly accurate, thus also eventually weakly accurate
  - $S \preccurlyeq P$
    - P is always strongly accurate, thus also always weakly accurate
  - $\Diamond S \preccurlyeq \Diamond P$
    - ◊P is always eventually strongly accurate, thus also always eventually weakly accurate



Trivial Reductions (2)

- Weakly complete
  - $\Diamond Q \preccurlyeq Q$
    - Q is always strongly accurate, thus also eventually strongly accurate
  - $\Diamond W \preccurlyeq W$
    - W is always weakly accurate, thus also eventually weakly accurate
  - $W \preccurlyeq Q$
    - Q is always strongly accurate, thus also always weakly accurate
  - $\Diamond W \preccurlyeq \Diamond Q$
    - ◊Q is always eventually strongly accurate, thus also always eventually weakly accurate



Completeness "Irrelevant"

- Weak completeness trivially reducible to strong
- Strong completeness reducible to weak
  - i.e. can get strong completeness from weak
    - $P \preccurlyeq Q$, $S \preccurlyeq W$, $\Diamond P \preccurlyeq \Diamond Q$, $\Diamond S \preccurlyeq \Diamond W$,
  - They' re equivalent!
    - $P \simeq Q$, $S \simeq W$, $\Diamond P \simeq \Diamond Q$, $\Diamond S \simeq \Diamond W$

Completeness \ Accuracy	Strong	Weak	Eventual Strong	Eventual Weak
Strong	P	S	◊P	◊S
Weak	Q	W	◊Q	◊W

49

Proving Irrelevance of Completeness

- Weak completeness ensures
  - Every crash is eventually detected by some correct node
- Simple idea
  - Every node q broadcasts suspicions **Susp** periodically
  - upon event receive <**S**,q>
    - Susp := (Susp ∪ S) – {q}

also works like a heartbeat

- Every crash is eventually detected by all correct p
  - Can this violate some accuracy properties?

50

Maintaining Accuracy

- Strong and Weak Accuracy aren't violated

- Strong accuracy
  - No one is ever inaccurate
  - Our reduction never spreads inaccurate suspicions

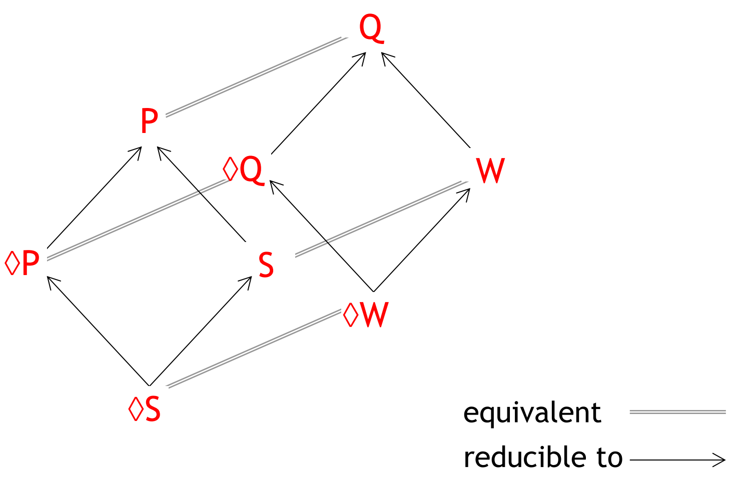
- Weak accuracy
  - Everyone is accurate about at least one node p
    - No one will spread inaccurate information about p

Maintaining Eventual Accuracy

- Eventual Strong and Eventual Weak Accuracy aren't violated

- Proof is almost same as previous page
  - Eventually all faulty nodes crash
  - Inaccurate suspicions undone
    - Will get heartbeat from correct nodes and revise (-{q})

Relation between FDs



Omega also a FD

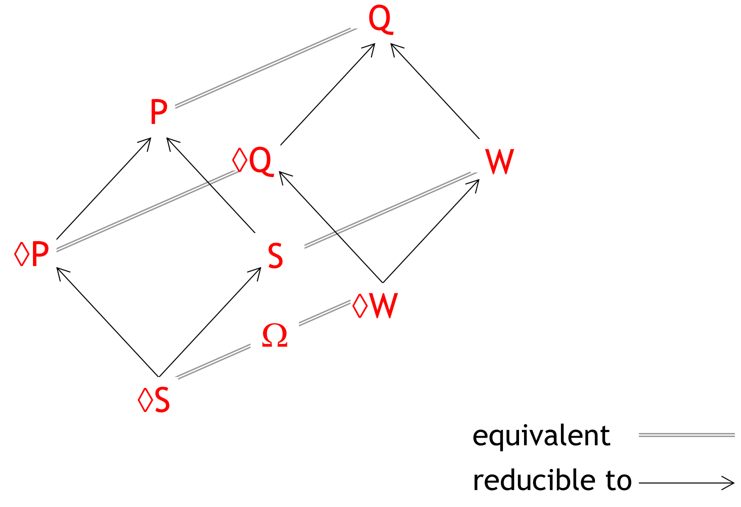
- Can we implement ◊S with Ω? [d]
  - I.e. is it true that ◊S≼Ω?
  - Suspect all nodes except the leader given by Ω
    - Eventual Completeness
      - All nodes are suspected except the leader (which is correct)
    - Eventual Weak Accuracy
      - Eventually, one correct node (leader) is not suspected by anyone
  - Thus, ◊S≼Ω

Ω equivalent to ◊S (and ◊W)

- We showed ◊S≼Ω, it turns out we also have Ω≼◊S
  - I.e. Ω≃◊S
- Due to the famous CHT Theorem
  - CHT Theorem (1996):
    - If consensus implementable with detector D, then Omega can be implemented using D
  - I.e. if Consensus≼D, then Ω≼D
    - Since ◊S can be used to solve consensus, we have Ω≼D
  - Implies ◊W is weakest detector to solve consensus
  - Only important proof that is omitted in this course!

55

Relation between FDs (2)



56

Combining Abstractions

Crashes
+
How they are detected

- **Fail-stop (synchronous)**
  - Crash-stop process model
  - Perfect links + Perfect failure detector (P)
- **Fail-silent (asynchronous)**
  - Crash-stop process model
  - Perfect links
- **Fail-noisy (partially synchronous)**
  - Crash-stop process model
  - Perfect links + Eventually Perfect failure detector (◊P)
- **Fail-recovery**
  - Crash-recovery process model
  - Stubborn links + …

57

Rest of course: “from P to ◊P”

- Assume crash-stop system with a perfect failure detector (fail-stop)
  - Give algorithms
- Then try to make a weaker assumption
  - For example, eventually perfect failure detector
  - Revisit the algorithms:
    - Do they still work? Can they be made to work?

58