Exam Master IFI, Polytech SI5, CSSR and Ubinet 2011/2012 All documents allowed

F. Baude and al.

Exercise 1 (5 points)

You surely remember the algorithm you were asked to provide in order to count the number of processes in a given connected graph of processes, where each edge between processes represents the fact that the communication channel is bi-directional.

You also may have noticed that the key point was to avoid to count more than once a given process, because the graph may contain some cycles. To do so, the simplest for any process was to send back an Echo message to a process that sent you a Probe message, containing a value of 0, in case you already received such a probe from another process.

This exercice is to take this as a starting point, in order to devise a solution in order to **count the number of edges** in the graph. As previously, you assume that the computation is initiated from a given process, called initiator.

Provide an algorithm, described using the style of programming we are used to (set of code blocks organized as guard -> actions). Thanks to a simple example graph owing at least one cycle, explain why your solution solves the problem.

Exercise 2 (2 points)

We can read in the 2nd course about Cristian's method for synchronizing the physical clocks between any two machines P and S (P willing to synchronize its own clock with respect to the one available at the source S):

Accuracy is ±(RoundTrip / 2 - min) whenever we know ***min****, the minimum delay for a one-way communication between P and S.* Notice that min applies for a message travelling in either way, i.e. from P to S, or from S to P. Remember that P sets its own clock, thanks to the value "t" appearing in the message, to the value: t + RoundTrip/2, where "t" corresponds to the clock value read at S and transmitted back to P in a message. RoundTrip is the measured delay at P for its request to reach S, measure the clock value at S i.e. "t" (measurement assumed to be instantaneaous), and comes back at P. Accuracy pertains to the difference between the two physical clocks, on P and on S, as the result of the P' clock synchronization process, i.e. t(P) - t(S).

The goal of the exercise is to demonstrate the claim made above about the Accuracy, ie that Accuracy is ±(RoundTrip / 2 - min).

To help you, we suggest you proceed in steps as follows:
1)Explain why the time on S (noted t(S)) is in the following range when the message with

the value "t" is received on P

$$t(S) \in [t + min, t + RoundTrip - min] \tag{1}$$

2) Using the fact demonstrated at step 1, conclude that the value set for P' clock, t(P)=t + RoundTrip/2, is such that t(P)-t(S)=±(RoundTrip / 2 - min)

Exercise 3 (7 points)

Question 1: Give an example of an execution of Dijkstra's self-stabilizing token ring for N = K = 4, with the following arbitrary variables state : $X_0 = 1$, $X_1 = 2$, $X_2 = 3$, $X_3 = 4$.

- Draw the various rings with the variables state, and draw the tokens within the ring.
- What are the properties that such an algorithm has to ensure ? Show me in your rings that they are satisfied.

Question 2: What are the properties of a Uniform Consensus ? Classify them into Safety / Liveness properties and explain me, in your own words, what they are used for.

In Figures 1 and 2 you have two consensus algorithms.

- Tell me the main difference between these two algorithms, with respect to the properties that they should satisfy. Classify the property in terms of liveness and safety.
- Draw two execution diagrams explaining this difference.
- What are the message and time complexities (in communication steps) of both algorithms? Argue with your own words.

Question 3: What are the properties of an Eventually Perfect Failure Detector ? Classify them into Safety / Liveness properties and explain me, in your own words, what they are used for.

- Compare a Failure Detector with a Leader Election, what are they used for, explain the differences with respect to the properties that they should satisfy. Classify the property in terms of liveness and safety.
- Does the following statement satisfy the synchronous processing assumption: " on my gaming server, no request takes more than 1 day to be processed " ?
- Is it possible to implement an Eventually Perfect Failure Detector in an asynchronous network? If so, explain how. If not, explain why not.

Exercise 4 (6 points)

This exercise tries to study alternative strategies to coordinate checkpoints. The objective is to see whether placing checkpoints at predefined points is sufficient to ensure the existence of coherent cuts. In particular, CIC (communication induced checkpointing) protocols require to

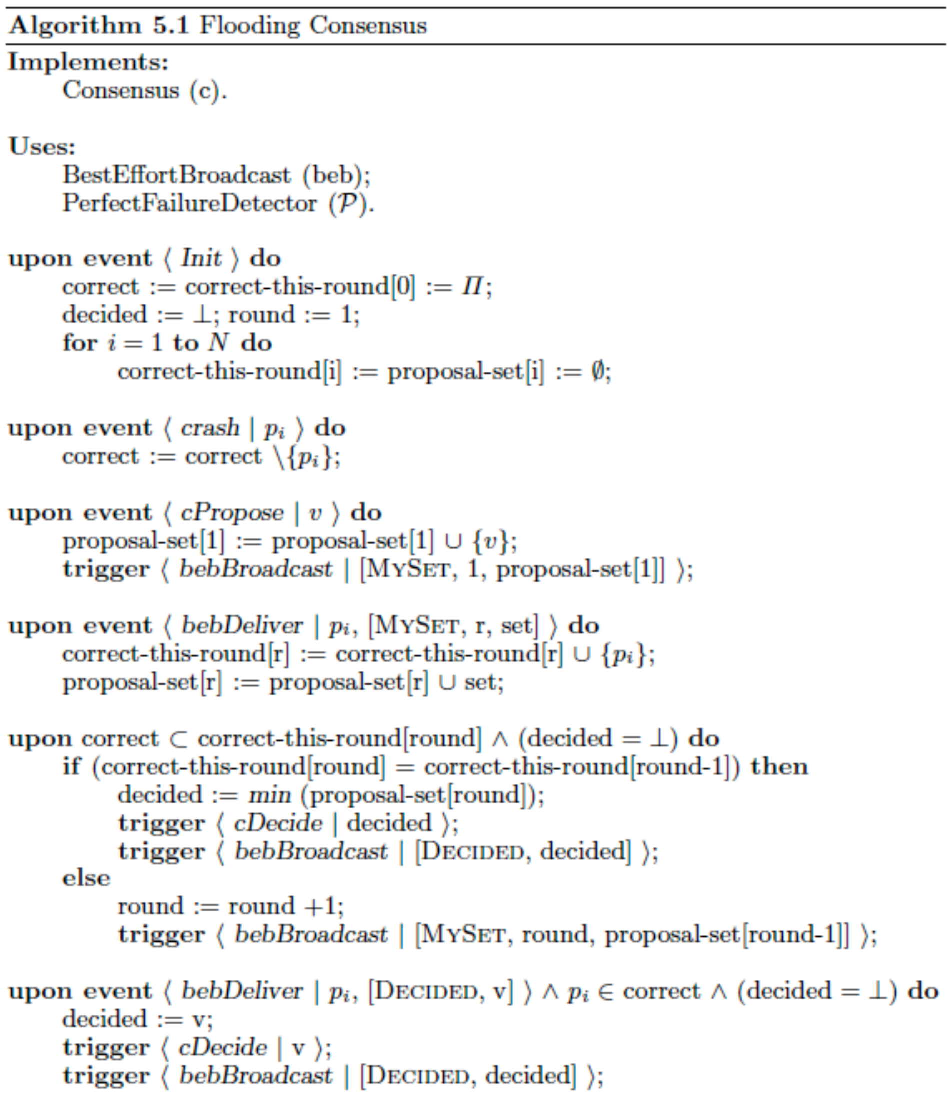
```
Algorithm 5.4 Hierarchical Uniform Consensus
Implements:
    UniformConsensus (uc).

Uses:
    ReliableBroadcast (rb);
    BestEffortBroadcast (beb);
    PerfectPointToPointLinks (pp2p);
    PerfectFailureDetector ($\mathcal{P}$).

upon event ⟨ Init ⟩ do
    proposal := decided := $\bot$; round := 1;
    detected := ack-set := $\emptyset$;
    for i = 1 to N do proposed[$i$] := $\bot$;

upon event ⟨ crash | $p_i$ ⟩ do
    detected := detected $\cup$ { $rank(p_i)$ };

upon event ⟨ ucPropose | $v$ ⟩ $\wedge$ (proposal = $\bot$) do
    proposal := $v$;

upon (round = $rank$(self)) $\wedge$ (proposal $\neq \bot$) $\wedge$ (decided = $\bot$) do
    trigger ⟨ bebBroadcast | [PROPOSE, round, proposal] ⟩;

upon event ⟨ bebDeliver | $p_i$, [PROPOSE, r, v] ⟩ do
    proposed[r] = v;
    if r $\geq$ round then
        trigger ⟨ pp2pSend | $p_i$, [ACK, r] ⟩;

upon round $\in$ detected do
    if proposed[round] $\neq \bot$ then
        proposal := proposed[round];
    round := round + 1;

upon event ⟨ pp2pDeliver | $p_i$, [ACK, r] ⟩ do
    ack-set := ack-set $\cup$ {$rank(p_i)$};

upon |ack-set $\cup$ detected| = $N$ do
    trigger ⟨ rbBroadcast | [DECIDED, proposal] ⟩;

upon event ⟨ rbDeliver | $p_i$, [DECIDED, v] ⟩ $\wedge$ (decided = $\bot$) do
    decided := v;
    trigger ⟨ ucDecide | v ⟩;
```

Figure 1: Hierarchical Uniform Consensus

be able to place a checkpoint *before the reception* of some messages; as the running platform may not authorize such an intrusion to the communication framework, let us study how we could place checkpoints upon the sending of messages, which is in general much easier to control (the programmer triggers himself the message sending).

Question 1: a first attempt Suppose we first design a checkpointing algorithm performing a checkpoint *before the sending* of each message:

- Draw on Figure 3 (last page of the exam) the checkpoints as specified by the algorithm
- Draw on Figure 3 the rollback dependency graph



Algorithm 5.1 Flooding Consensus

```
Implements:
    Consensus (c).

Uses:
    BestEffortBroadcast (beb);
    PerfectFailureDetector ($\mathcal{P}$).

upon event ⟨ Init ⟩ do
    correct := correct-this-round[0] := $\Pi$;
    decided := $\perp$; round := 1;
    for $i = 1$ to $N$ do
        correct-this-round[i] := proposal-set[i] := $\emptyset$;

upon event ⟨ crash | $p_i$ ⟩ do
    correct := correct $\setminus \{p_i\}$;

upon event ⟨ cPropose | $v$ ⟩ do
    proposal-set[1] := proposal-set[1] $\cup$ $\{v\}$;
    trigger ⟨ bebBroadcast | [MySet, 1, proposal-set[1]] ⟩;

upon event ⟨ bebDeliver | $p_i$, [MySet, r, set] ⟩ do
    correct-this-round[r] := correct-this-round[r] $\cup$ $\{p_i\}$;
    proposal-set[r] := proposal-set[r] $\cup$ set;

upon correct $\subset$ correct-this-round[round] $\wedge$ (decided = $\perp$) do
    if (correct-this-round[round] = correct-this-round[round-1]) then
        decided := min (proposal-set[round]);
        trigger ⟨ cDecide | decided ⟩;
        trigger ⟨ bebBroadcast | [Decided, decided] ⟩;
    else
        round := round +1;
        trigger ⟨ bebBroadcast | [MySet, round, proposal-set[round-1]] ⟩;

upon event ⟨ bebDeliver | $p_i$, [Decided, v] ⟩ $\wedge$ $p_i \in$ correct $\wedge$ (decided = $\perp$) do
    decided := v;
    trigger ⟨ cDecide | v ⟩;
    trigger ⟨ bebBroadcast | [Decided, decided] ⟩;
```

Figure 2: Flooding Consensus

- Suppose the process 1 crashes as indicated on the figure; what is the best recovery line?
- Conclude: is this algorithm efficient? is it useful? can a domino effect occur with this algorithm?

- Explain: Considering the definition of a consistent cut, and of an orphan message, explain the answer to the preceding question.

Question 2: second attempt Suppose we now design a checkpointing algorithm performing a checkpoint *after* the sending of each message:

- Draw on Figure 4 the checkpoints described by the new algorithm
- Draw on Figure 4 the rollback dependency graph
- Suppose the process 1 crashes as indicated on the figure; What is the best recovery line?
- Conclude: is this algorithm efficient? is it useful? can a domino effect occur with this algorithm?
- Explain: explain why this algorithm is better than the previous one (you should again rely on the definition of consistent cuts / orphan messages)
- Analysis: As explained in the introduction, the main advantage of this algorithm is that it requires less control on the communication framework than CIC protocols. But what is the main drawback of this algorithm compared to CIC protocol?
- Bonus question: How many checkpoints have to be kept for each process? why? Said another way: is it possible to garbage collect some of the checkpoints and how?

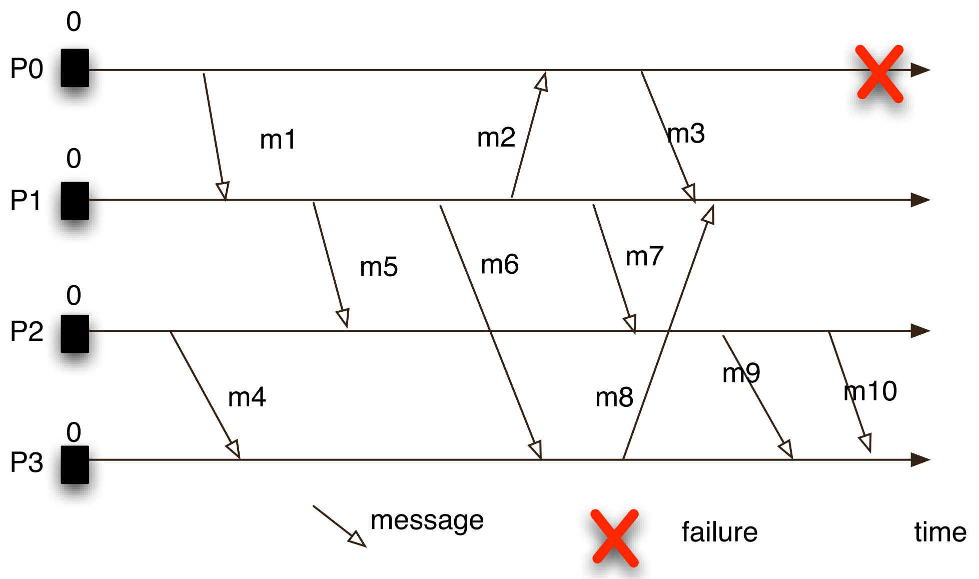


Figure 3: An execution with some messages: **Question 1**

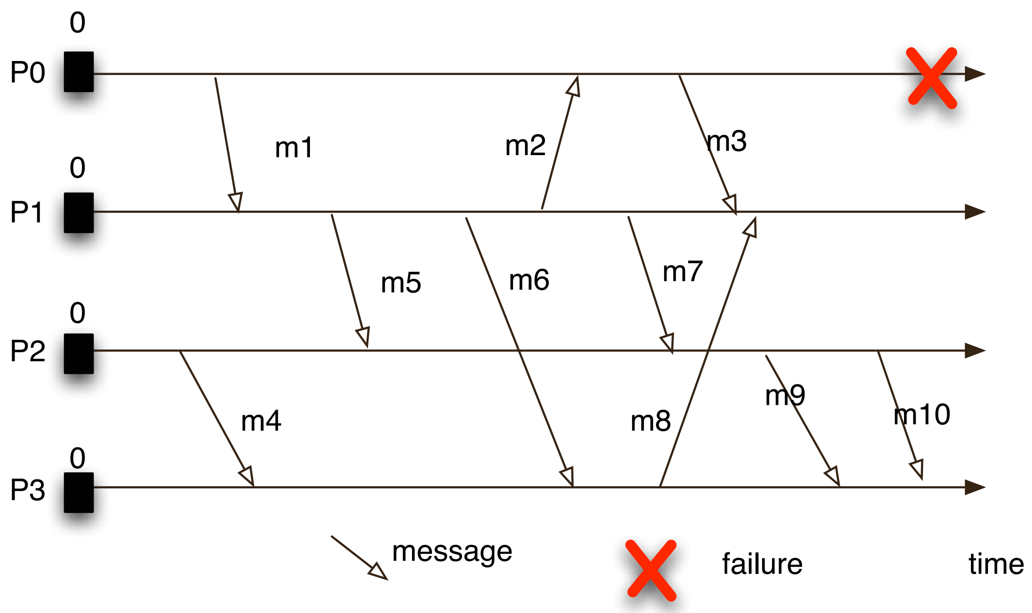


Figure 4: An execution with some messages: **Question 2**