Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik
Prof. Dr. W.-P. de Roever
Martin Steffen, Immo Grabe

# Verteilte Algorithmen

Wintersemester 2003/04

## Serie 6

1. Dezember 2003

**Thema :** — **Mittsemestertest! — (Aufgaben mit Lösungshinweisen)**

**Ausgabetermin: 1. Dezember 2003**

**Abgabe: 9. Dezember 2003**

**Mittsemestertest:** Diese Aufgabenserie ist von jedem *allein* zu bearbeiten! Die Aufgaben sind diesemal etwas breiter gestreut, d.h., auch 2 Aufgaben aus vorangegangenen Kapiteln sind mit aufgenommen. Bitte beachten Sie auch den Abgabetermin. Viel Erfolg!

Der Zettel behandelt *verteilte Einigung,* speziell die *randomisierte Variante*, und zwar in synchronen Netzen mit unzuverlässiger Kommunikation (Nachrichtenverlust).

**Aufgabe 1 (Kantengewichte und Kantenidentifikatoren (4 Punkte))** Bearbeiten Sie Aufgabe 4.18 aus [**?**].

**Aufgabe 2 (Byzantinismus (6 Punkte))** Bearbeiten Sie Aufgabe 6.21 über *EIGByz* aus [**?**].

**Solution:** The exersize is indended to get a feeling about how the EIG-algorithm for Byzantine failures works by playing around with scenarios, where the number of failed processors is large enough to undermine consensus.

1. *7 nodes, 2 failed processors, 2 rounds:* Since the algorithm's decision is based on *EIG-trees,* we firt recapitulate this structure to fill it with values which lead to *disagreement.*

   As usual for consensus algorithms, the basic idea is to send around values often enough to come to some conclusion based on majority. Since there might be Byzantine process failures, it is not enough to take the value received by a process "for the word of it" but one needs to hear a "second opinion" (and a third ... ) about the value of a process. In other words, each process not just remembers the value heard *directly* from a process, but also what another process heared the value would be, and this level of indirection must be deep enough to filter out the cheater. What makes it hard is, that a lyar may lie not only about his own value, but he can also lie about which value he knows about others, which creates confusions. This need of higher-order indirect information about

values, which forms the core of the exponential information gathering idea, makes the difference between the consensus problem in the presence of *stopping* failures and the situation here. When only stopping failures occur, needs no higher levels of indirection.[1]

May it as it is, the information of one player "process $i$ said value $v$", "process $j$ said, that process $k$ said that value $v'$" ... is stored in a hierarchical data structure, the EIG-tree, whose depth is given by the number of rounds. In this exercise, we play $f = 2$ rounds which gives $f + 1 = 3$ level.[2] Each level in the tree, counted from the root, corresponds to the level of indirection of the information, so nodes, say, 2 and 23, correspond to "first-hand information" and "second-hand information" of some process about node 2. In the first round, located at level one in the tree, 2 directly gave its value, and in the second round resp. at the second tree level, 3 relayed the information that 2 possess some given value.

In our example, each tree —there are 7— consists of 7 nodes in the first level each of which possess 6 decendants in the next level, which form the leaves. To systematically approach construct a *disagreement* situation, let us consider the consequences of the 5 honest vs. 2 lying processes situation in terms of the given algorithm. If a sound process broadcast some value, say 0 in the first round, then only two of the values may arrive *forged* as second-level information at the processes. This means, at the second level of each tree, the leaves contain at least *4* original values about that process, which makes a majority of the 6 leaves.

According to the decision procedure of the *EIGByz*-algorithm and the decision tree, it means that the father-node of the 6 leaves is labelled with an *accurate* opinion about the picked process. Taking the 5 nodes of level 1, corresponding to sound processes, this means that in the decision tree[3] at level one there a *5 correct labels*. And this for all sound processes.

To lead this to a contradiction, we therefore need *at least two different start values*. say *true* and *false*, in 3 : 2-ratio.[4] To be concrete, let the processes $1, \ldots, 5$ be sound and 6 and 7 the cheaters and let furthermore *false* be the initial value of 1, 2, and 3, and 5 and 6 are initially *true*.

As mentioned, this fixes 5 values at level 1 of the trees of all non-cheaters to the array:

$$\text{for all } T_i \text{ with } i = 1, \ldots, 5: \quad \mathit{false}, \mathit{false}, \mathit{false}, \mathit{true}, \mathit{true}, ?, ? \tag{1}$$

Now we need the cheaters to fill in the question marks of two different non-cheaters in such a way that they decide at their respecive root node in a different way.

Consider, for instance, the cheater 6, respecively the tree node 6 at level 1 in the trees of the honest processes. In order than some of the sound processes considers 6 to have value *false* instead of the ?, there must be a majority of leaves that has heared about

[1] Since messages can be lost, one cannot directly base the decision on what one heard from another process, because one could have heard nothing, while other players could have received a dissenting vote from the failing (=stopping) processes and thus would fall back to the default vote. That was the idea of the various algorithms for stopping failures.

[2] We start with "zero-knowledge" before the first round.

[3] of course again for a sound process, only

[4] One could have guessed that from the start, too.

*true*. Indeed, *three nodes* suffice, since the cheater 7 may lie about what he had heard about 6, filling in a 4th *false*.

As, independant of the node 67, there are already 3 *false*'s in the 6 leaves, there can't be a majority of *true* for all sound trees, independant of any further cheating of 7, the dissenting vote can only be *the default value.* To achieve this, at least 3 sons of tree node 6 are marked with *true*. This means, 6 must send

To sum up concretly: The cheaters and non-cheater processes are numbered as above and possess the start values of Equation (**??**). The disagreement is constructed by having conflict between 1 and 2. Cheater 6 sends *false* to $1, 2, 3$ and *true* to the rest in the first round; in the second round 7 *sends to* 1 *that it had heared that* 6 *is false*. This gives for the tree in process 1 that

| | |
|---|---|
| 61 | *false* |
| 62 | *false* |
| 63 | *false* |
| 64 | *true* |
| 65 | *true* |
| (66) | – |
| 67 | *false* |

Propagating this majority to the father 6 yields a *forth false* at level 1, and process 1 decides to *false*.

For process 2, the situation is harder since we need not just one *false* but we must avoid to assing another false to one of the ?, i.e., for node 6 and 7. The situation for the leaves $6j$ is already quite fixed, and the only place we still have freedom is the underline value for 67, which blocks the majority and leads to the default decision at node 6 in the tree for process 2. Remains the subtree for 7, where we also need to prevent a *false*-labelling. That's simple: 7 just sends *true* in the first round, and the other remaining cheater correctly states (for instance) that he had heard *true* from 7.[5]

| | | | |
|---|---|---|---|
| 61 | *false* | 71 | *true* |
| 62 | *false* | 72 | *true* |
| 63 | *false* | 73 | *true* |
| 64 | *true* | 74 | *true* |
| 65 | *true* | 75 | *true* |
| (66) | – | 76 | *true* |
| 67 | *true* | (77) | – |

**Aufgabe 3 (Ablauf eines I/O Automaten (4 Punkte))** Bearbeiten Sie Aufgabe 8.4 aus [**?**].

**Solution:** The excercise is to describe the (fair) traces of an automaton and especially the traces of a product automaton.

[5] As we have made no assumptions so far on the behavor of 7 except that it sends a value of "6 is *false*" to 1 and "6 is *true*" to 2 in the second round, we have much freedom here.

Figure 1: Automata $A$ and $B$

- *Behavior of A:* The automaton has only two states, and the only thing it can do is do a *go* step. Thus $[\![A]\!]^{trace} = \{\lambda, go\}$. The fact that $A$, as all I/O automata, is input-enabled, has no effect, since there are no input actions at all. For $A$'s fair traces we have to take care for the fairness condition of finite traces, namely that actions from each the task are not enabled. In our case, *go* must not be enabled, which removes the empty trace as unfair finite trace, yielding $[\![A]\!]^{fair} = \{go\}$.
- *Behavior of B:* This gets a bit more tricky.
  - Let's start with the ordinary traces. In the traces, the state (containing the natural number) are not recorded, and neither are the internal increment actions. One could write it as follows:

    $$go\ ack^*\ go^{\leq\omega} + \lambda\ .$$

    In the regular expression, $go^{\leq\omega}$ stands for $go^* + go^\omega$.
  - For the fair traces, we first observe that *go* is an input, and thus beyond control of the automaton.[6] This means there won't be an requirements concerning the *go*-action.

    The tasks of $B$ separate between the internal increment action *inc* and the *ack* action. Consider an infinite execution of $B$. $inc^{\leq\omega}$

    $$[\![B]\!]^{fair} = \lambda + go\ ack^* go^{\leq\omega}$$

- $A \times B$: For the product, we must combine the remove the *go*-inputs of $B$ from the inputs, nevertheless, *go* is still output, due to $A$. Furthermore, $B$'s output *ack* is common output. Note thet $A \times B$ has therefore empty input, it is not triggered from outside. Furthermore, $B$ does not received infinite input as before, when it was alone, but only the only *go*-action that its partner sends.
  - For the traces, this gives

    $$[\![A \times B]\!]^{trace} = \lambda + go ack^*$$

  - The fairness requirement resolve in the start location $s, on, i)$ the systematic preference of the internal increment operation: since go is constantly enabled, it must be taken at some point. This gives as possible fair traces:

    $$[\![A \times B]\!]^{fair} = go\ ack^*$$

**Aufgabe 4 (Traces und fair traces (6 Punkte))** Bearbeiten Sie Aufgabe 8.12 aus [**?**].

**Solution:** First we restate the properties:

[6] As required from I/O-automata, $B$ is input-enabled wrt. *go*. Furthermore, of course, *go* is not mentioned in the task-part of the automaton.

**Lemma 1 (Safety)** Let $P$ a safety property and $A$ an I/O-automaton. Then the following 3 statements are equal:

1. $[\![A]\!]^{trace} \subseteq [\![P]\!]^{trace}$
2. $[\![A]\!]^{fair} \subseteq [\![P]\!]^{trace}$
3. $[\![A]\!]^{fin} \subseteq [\![P]\!]^{trace}$

**Proof:** Since obviously each finite and each fair trace is, a fortiori, a trace, two implications, (1) $\rightarrow$ (2) and (1) $\rightarrow$ (3), are straightforward by transitivity of set inclusion. Let's concentrate on the rest, where we show two to close the circle.[7]

(3) $\rightarrow$ (1)**:** So assume (3) and additionally $\alpha \in [\![A]\!]^{trace}$. If $\alpha$ is finite, we are done by assumption. If $\alpha$ is infinite, then consider $\alpha_i$, the prefixes of length $i$ of $\alpha$. All of them are finite. Furthermore, $\alpha_i \in [\![A]\!]^{trace}$ for all prefixes,[8] which yields $\alpha_1 \in [\![P]\!]^{trace}$, whence, by limit closure of the safety property $P$, the result follows.

(2) $\rightarrow$ (3)**:** This one is a bit more tricky, and we use the fact, that each *finite* trace of an automaton can be extended to a fair one (cf. Lemma **??**).

So assume (2) and additionally some $\alpha \in [\![A]\!]^{fin}$. By the mentioned Lemma **??**, there exists an extension $\alpha'$ of $\alpha$ s.t., $\alpha' \in [\![A]\!]^{fair}$, and thus by assumption, $\alpha' \in [\![P]\!]^{trace}$. Using another closure condition of the safety property $P$, this time *prefix closure,* gives $\alpha \in [\![P]\!]^{trace}$, as required.

□

[7]Some proved the following implications using the contrapositive formulation. We find that slightly less transparent, but of course the idea is the same.

[8]The fact, that the traces of an automaton are prefix closed seems trivial, of course, but is a crucial step in the argument. What's behind is that, when considered as a *trace property*, $[\![A]\!]^{trace}$ is a safety property. We only need prefix closure from that observation here.