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 17. Dezember 2003

Serie 7

Thema : **(Aufgaben mit Lösungshinweisen)**

Ausgabetermin: 17. Dezember 2003

Abgabe: 5. Januar 2004

Aufgabe 1 (Automatenkomposition (6 Punkte)) Beweisen Sie Theorem 8.3 aus [1, Seite 211].

Solution: The theorem is more or less direct consequence of Theorem 8.2.

Let β be some sequence of external actions of $A = \prod A_i$. If $\beta \downarrow_{A_i} \in [\![A_i]\!]^{trace}$, then $\beta \in [\![A]\!]^{trace}$.

Let β_i abbreviate $\beta \downarrow_{A_i}$. By definition, $\beta_i = \mathit{trace}(\alpha_i)$ for some local execution $\alpha_i \in [\![A_i]\!]^{exec}$. Thus by the composition property for *executions* (Theorem 8.2 of [1]), there exists a global execution $\alpha \in [\![A]\!]^{exec}$ such that $\alpha_i = \alpha \downarrow_{A_i}$ and $\beta = \mathit{trace}(\alpha)$. By definition, $\beta \in [\![A]\!]^{trace}$, as required.

What's interesting besides the question in this exececise could be, why underlying Theorem 8.2 does *not* hold if one gives up the assumption that only *finitely many* processes are *synchronizing* in a step. The theorem is a "composition property" for *traces:* does the existance of local traces imply the existance of a global trace? In the presence of infinite synchronization, this conclusion fails. □

Aufgabe 2 (Fairness (6 Punkte)) Beweisen Sie Theorem 8.7 aus [1, Seite 215].

Solution:
Bemerkung: 2 + 1 + 2 + 1

1. The mentioned lemma highlights the *core* of the fairness definition. In Lynch, fairness of an execution (and indirectly of a trace) has two clauses, namely a criterion for a finite trace and one for an infinite. The lemma can be interpreted that the important part is the clause for the infinite trace,[1] since if an automaton exhibits an unfair finite trace, the machine can always continue in a fair way; in this way, finite unfairness is not very

[1]In effect, other people define fairness differently, perhaps leaving out a criterion for finite traces. But no one leaves out a clause for infinite traces, since this is the part that counts. Apart from that, one can define *other*, harder conditions on the infinite traces.

important. The property is furthermore interesting as it has some *constructive* content. It states that fairness can be achived by choosing appropriately among enabled tasks. The piece of program which performs this choice is usually known as the *scheduler.* The lemma is listed as *Theorem 8.7* in [1, p. 215].

Lemma 1 (Finite unfairness is not forever) For each *finite* execution α of A, there exists a *fair* extension of A. The same holds analogously for traces.

If α is a finite executions of A and β a (finite or infinite) sequence of *input actions* of A, then there is a fair execution $\alpha \cdot \alpha'$ such that the sequence of input actions in α' is β.

If β is a finite trace of A and β' a (finite or infinite) sequence of *input actions* of A, then there is a fair execution $\alpha \cdot \alpha'$ such that $\mathit{trace}(\alpha) = \beta$ and such that the sequence of input actions in α' is β'.

Proof: executions Here's some proof sketch for *executions.* The crucial idea is that, given a trace, either the trace is fair, then we're done, or if not, we can make in longer one step and re-judge its fairness. Either this process stops after a finite number of steps, and we have the desired result. Or it does not stop,[2] in which case the criterion for infinite executions tells us that at least the infinite trace is fair.

A bit more precise: We are given the automaton A and furthermore C as the task partitioning on the locally controlled actions of A. Furthermore we start with a finite execution α_0.

(a) if none of the classes C is enabled in the final state of α_0, we are done, the trace is fair.

(b) Otherwise, there exists a class C which is enabled; so α_0 is not fair. But since there is some actions enabled, the hope is not lost, we can continue.[3]
Now we need to be a bit careful, choosing the next action, since we do not just wish to construct an infinite execution, but an infinite *fair* execution. Therefore, if there is more than one task to choose from we must avoid to systematically ignore one.[4] What we need to implement is obiously a (fair) *scheduler.* For instance, if there are, at the current state, the tasks $C_1, \ldots\ C_n$ enabled, we could just take an action of the task which has been pending the longest time, i.e., continuously enabled without being taken. That's some sort of FIFO strategy.

(c) This gives execution α_1, and we apply the same criterion.

If this process does *not* stop, we obtain an infinite trace, and the strategy assures fairness!

[2]One could say it stops at time ω, the first infinite ordinal, if one prefers that :-)

[3]Remember: a task is enabled if some of its (internally controlled) actions is enabled, i.e., can be taken by the automaton. Internally controlled actions refer to those the automaton can perform autonomously, namely internal actions and output actions.

[4]Remember that the crucial condition for fairness, i.e., the one for infinite executions, states that for all tasks C: if there are finitely many occurrencies where C is *disabled*, then there are infinitely many occurrences of events from C. Finite many occurences of disabledness means, that C, from some point on, is enabled forever until eternity.

traces As the notions of traces and fairness for traces are derived ones, this implies the corresponding property for traces, as well. So assume a finite trace β; by definition, the finite trace is caused by some *execution* α of A, which a priori might be infinite. Since the only way that the execution is infinite, is that there is an infinite sequence of internal actions "at the end", there is also a *finite* execution $\alpha' \preceq \alpha$ with $\beta = \mathit{trace}(\alpha')$. By the corresponding properties for executions, the finite α' can be extended to a *fair execution* α'' such that $\alpha' \preceq \alpha''$. Since the projection onto traces is monotone wrt. the prefix ordering, we immediately get $\beta = \mathit{trace}(\alpha') \preceq \mathit{trace}(\alpha'')$, which gives the desired result, since $\mathit{trace}(\alpha'')$ is a fair trace by definition.

input executions The property that an given execution can be extended in a fair way by fixing even the input action is proven simularly as first property. The fact that I/O automata have this property hinges on two facts: First, the automata are *input enabled,* i.e., an auotmaton cannot refuse an input. Second (and of course related to the first point), fairness imposes no restrictions on the input actions, but only for the internally controlled ones.

More concretely: We start with a finite execution $\alpha \in [\![A]\!]^{exec}$. As in part 1 of the theorem, we show a strategy for a scheduler, which works of β one after the other. If β is infinite, one possible construction works as follows:

- first we do the next input from $\beta = a.\beta'$.
- Afterwards we pick from the enabled tasks (if any) the one which has been enabled for the longest period in time up-to know but not been served. If no task is enabled we skip this step. We continue with β'.

This process never ends, since β is infinite. Since the number of tasks is finite, none is neglegted for ever.

If β is finite, then the mentioned process stops after working off β completly. By part (1) of the theorem we can extend the execution into a fair one, as required.

input trace □

For the lemma: remember that *safety* (and liveness) have some *higher-order* flavor. While fairness is a property of a trace, *safety* is not a property of a trace, it a property of a set of traces (in other words it's a property of (trace) properties).

Remark 1 (Infinity and fair traces) Last week we had some discussion about (countable) infinity, especially in connection with fair strategies and the number of tasks. The I/O automata as defined in [1] allow a countably infinite number of tasks. I claimed the following:

> The properties about fair extensions of Lemma 1 hold only for a finite number of tasks.

What was my argument? Consider an automaton with a countable infinite number of tasks.[5] Since the number is countable infinite, they can be enumerated (that's the meaning of "countable"): $t_0, t_1, t_2, \ldots$. Assume further that always at least one action in each task is enabled.

[5]The tasks are equivalence classes of internally controlled actions. Since fairness does not require that *particular* actions have to be taken in the long run, just some action per task, the question whether the number of actions within the tasks are finite or infinite, is not important for the discussion.

So always, each task can do something. What kind of behavior results from the strategy sketched above? Well, picking always (one of) the tasks which has been enabled but not taken for the longest time just takes an action from each of the tasks one after the other, for instance in the order of the enumeration: $t_0, t_1, t_2 \ldots$. This, however, is an *unfair* execution, since, for instance, t_0 is enabled an infinite amount of time after it has been taken, but not taken a second time; the same applies to the other tasks.

Is thus the Lemma wrong? No, it just states that the above stategy does not extends a finite execution to a fair one.

Thinking about what went wrong with the above strategy can help to find a strategy, which can handle infinitely many tasks, as well. The problem, in some sense, was the infinite reservoir of longest-outstanding tasks allows that a task already served once to be postponed forever.[6] To repair it, we must change the strategy; picking longest unserved one is natural for finite many tasks but unfair for infinite many.

Given the tasks $t_0, t_1, t_2, \ldots$, assume the following enumeration:[7]

$$t_0, t_0, t_1, t_0, t_1, t_2, t_0, t_1, t_2, t_3, \ldots$$

It has the property that each task appears infinitely many times; note also that also each suffix has this property (not by coincidence, by the way). The scheduler then does the following: it works down the list one after the other. At each step, it takes the next one mentioned in the list which is enabled, and continues in the enumeration from there.

Note that, in the everyday, intuitive meaning of the word, the stategy looks "unfair" since it tends to take t_0 more often then t_1 etc. Well, nobody said that the mathematical definition of fairness is intuitive ...

Aufgabe 3 (Lossy Fifo (6 Punkte)) Bearbeiten Sie Aufgabe 14.4 von [1].

Solution:

Cf. also the universal non-lossy fifo buffer on page 204.

That's simple. One can either lose the message by not putting them into the queue, when the message is input, i.e., at the *send*-action. Or one can add a internal actions which loses a message in the queue. In principle, any message of the queue can get lost. Since the exercise requires that the channel has complete freedom to lose (or not to lose) messages, one must be careful not to impose unwanted *fairness restrictions* to the automaton. For instance, if one choses the solution with an additional internal lose-action, one must not put the output (= *receive*) actions into a different task than the lose-actions. Otherwise we force the channel to lose a message from time to time, and also prevent that all messages are lost. $\square$.

Wir wünschen fröhliche Weihnachten, einen guten Rutsch und ein erfolgreiches neues Jahr!

[6] until it's served "again", so to speak .... If there are only finitely many tasks, the problem is not present. If only finitely many tasks are enabled continously, then once a task is taken, it will be taken *again* later, namely when it has become the oldest, unserved one. Of course, a fair scheduler for infinitely many tasks is not too relevant from a practical point of view.

[7] As far as I remember, this technique of enumeration is called "dovetailing".

References

[1] Nancy Lynch. *Distributed Algorithms.* Kaufmann Publishers, 1996.