Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn
Juho Hirvonen

August 15, 2018

Network Algorithms, Summer Term 2018
Problem Set 10 – Sample Solutions

Exercise 1: Consensus and deterministic message passing

Note: we assume that the initial states are just the inputs of the processes and that the scheduler has a chance to crash a process before a message is sent. Without the latter assumption consensus could be solved. We could alternatively define that the output must be one of the inputs of the correct processes, but this would be consesus trivially impossible since after last messages have been sent the process with the input to be decided could be crashed, making the decision invalid.
Let x_1 and x_2 denote the inputs of processes p_1 and p_2, respectively. Let $S_1(t)$ and $S_2(t)$ denote the states of the processes after the first t choices of the scheduler, and let $S(t) = S_1(t) \cup S_2(t)$ denote the full state of the system.

a) **Initial state is bivalent.** Consider the state $S(0)$ for inputs $x_1 = 0$ and $x_2 = 1$. Now if wecrash p_2, the system must converge to output 0. On the other hand, if we crash p_1, then the system must converge to output 1. Therefore the initial state is bivalent.

b) **Critical state must exist.** Assume $S(t)$ is a state that is not critical, i.e., at least one of the choices of the scheduler leads to a bivalent state. Then the scheduler can always choose this state $S(t+1)$ and continue the execution. If there are no critical states, no matter what the scheduler does, it will always have a choice that continues the execution of the algorithm.

c) **A critical state does not exist.** By definition the state of the system includes at most two messages that have been sent, but have not yet been delivered. This is since initially there are two messages, and new messages are only sent when previous messages are received.

   Therefore at each state, the scheduler has two choices to make: either message m_1 or m_2 could be received. These messages could either be both sent by one process, or there could be one message sent by each process. Assume the first case with both messages having been sent by p_1, and assume that $S(t)$ is a critical state. Without loss of generality assume that scheduling m_1 first will produce a 0-valent state $S(t+1)$ and scheduling m_2 first will produce a 1-valent state $S'(t+1)$. This, however, is a contradiction since we can crash p_2 after either m_1 or m_2 arrives and p_1 cannot distinguish between the two cases, and must be able to produce the same set of outputs. Now consider the case where m_1 was sent by p_1 and m_2 by p_2. We can schedule either m_1 or m_2 first and then crash, say, p_2. Since p_1 cannot distinguish in which order the messages arrived, it must have the same same set of possible outputs in both cases.