XMP 2012/13 Final exam (theory part)

Verifying communications protocols

Consider a one-way communications link that may drop packets from time to time (but not reorder or corrupt them – a damaged packet will be silently discarded). A representative example could be a TV satellite downlink. Since there is no way to send back acknowledgments or requests for retransmissions, a simple technique for improving the reliability of the link is to proactively send every packet multiple times, with the expectation that at least one of the copies will be delivered. In order to distinguish between two copies of the same packet, and two consecutive packets that happen to have the same content, we also include in each packet a *sequence number*, so that the receiver can discard any subsequent packets with the same number.

We will look at a particularly simple instance of this scheme, where each packet is transmitted twice, and the sequence numbers count up modulo 2, i.e., we just distinguish between even-numbered and odd-numbered messages. For the purpose of the analysis, we will assume that the link is *semi-reliable*, in the sense that it will never drop two packets in a row.

We model this system as three CSP processes in a pipeline-like configuration. In the middle, the *LINK* process, with input channel xmit and output channel recv, is defined by:

$$\mathit{LINK} \triangleq \mathsf{xmit}?p \to ((\mathsf{recv}!p \to \mathit{LINK}) \sqcap (\mathsf{xmit}?p' \to \mathsf{recv}!p' \to \mathit{LINK}))\,.$$

Note how, if the link accepts a packet p for transmission, it may choose to either deliver it (and go back to its initial state), or silently discard it, but then definitely deliver the next one (p'). Though it doesn't particularly matter to the *LINK* process, the alphabets of both channels consist of all well-formed *packets*, i.e., pairs $\langle s, b\rangle$ where $s \in \mathbf{String}$ is an arbitrary string (the payload) and $b \in \{0, 1\}$ is the sequence number.

The transmitter is represented as a process XMIT_n, parameterized by the message number $n \in \mathbf{N}$, and with xmit as an output channel. The payload to be transmitted in the n'th message is given by some unspecified function $\mathit{data} : \mathbf{N} \to \mathbf{String}$. The transmitter constructs a packet consisting of the payload and the least significant bit of the message number, transmits it twice, and then increments the number and repeats:

$$\mathit{XMIT}_n \triangleq \mathsf{xmit}!\langle \mathit{data}(n), n\%2\rangle \to \mathsf{xmit}!\langle \mathit{data}(n), n\%2\rangle \to \mathit{XMIT}_{n+1}\,.$$

Finally, the receiver is represented as a process RECV_b, where $b \in \{0, 1\}$ is the currently expected sequence number. The process has input channel recv, and an output channel out, to which the receiver delivers the received payload strings (without the sequence numbers):

$$\mathit{RECV}_b \triangleq \mathsf{recv}?p \to ((\mathsf{out}!\mathit{fst}(p) \to \mathit{RECV}_{\neg b}) \lhd \mathit{snd}(p) == b \rhd \mathit{RECV}_b)\,.$$

Here, the functions $\mathit{fst}(\cdot)$ and $\mathit{snd}(\cdot)$ extract the first and second element of a sequence, respectively, i.e., $\mathit{fst}(\langle x, y\rangle) = x$ and $\mathit{snd}(\langle x, y\rangle) = y$. $\neg b$ is bit negation: $\neg 0 = 1$ and $\neg 1 = 0$; or $\neg b = (b+1)\%2 = 1-b$. The construct $P \lhd e \rhd Q$ is the CSP notation for if-then-else: if the value of the boolean expression e is *true*, the process behaves like P, otherwise like Q.

The receiver gets a packet p from the link. If the sequence number of the packet matches what the receiver is expecting, it delivers the payload on channel out, and waits for the next packet (with the expected sequence number updated); if the expected and actual sequence numbers don't match, the packet is simply ignored.

We construct the entire system as follows:

$$\mathit{SYS} \triangleq (\mathit{XMIT}_0 \parallel \mathit{LINK} \parallel \mathit{RECV}_0) \setminus \{\mathsf{xmit}, \mathsf{recv}\}\,.$$

That is, the system has just out as an output channel.

Verifying the protocol

From the informal description above, we might expect the system to correctly deliver all the packets, and this is indeed the case. Consider the following definition of the desired behavior of the system:

$$\begin{aligned} \mathit{OUT}_n &\triangleq \mathsf{out}!\mathit{data}(n) \to \mathit{OUT}_{n+1} \\ \mathit{DES} &\triangleq \mathit{OUT}_0 \,. \end{aligned}$$

Question 1. Show, using the CSP laws, that the system behaves exactly as desired:

$$\mathit{SYS} = \mathit{DES}\,.$$

Hint: You will need to be very careful and systematic in the derivation, or you are virtually guaranteed to make a mistake. Introduce explicit names (parameterized over the variables they contain) for all the program points of each process. Compute the net result of composing the two leftmost processes in the pipeline first, and simplify it as much as possible (including any applicable channel concealments) before composing it with the rightmost process.

A broken system

Consider now what would happen if, due to a coding error, or persistent hardware fault in the transmitter, only one copy of each packet is actually sent. That is, we define a broken transmitter process:

$$\mathit{BXMIT}_n \triangleq \mathsf{xmit}!\langle \mathit{data}(n), n\%2\rangle \to \mathit{BXMIT}_{n+1}\,,$$

but keep the other components unmodified, so that the alternate total system now looks like:

$$\mathit{ASYS} \triangleq (\mathit{BXMIT}_0 \parallel \mathit{LINK} \parallel \mathit{RECV}_0) \setminus \{\mathsf{xmit}, \mathsf{recv}\}\,.$$

Recall also the notion of a *failure* of a process:

$$\mathit{failures}(P) = \{(s, B) \mid s \in \mathit{traces}(P) \wedge b \in \mathit{refusals}(P\,/\,s)\}\,,$$

and the definitions of *trace-refinement* and *failure-refinement*:

$$\begin{aligned} P \sqsubseteq_{\mathrm{T}} Q &\Leftrightarrow \mathit{traces}(Q) \subseteq \mathit{traces}(P) \\ P \sqsubseteq_{\mathrm{F}} Q &\Leftrightarrow \mathit{failures}(Q) \subseteq \mathit{failures}(P)\,. \end{aligned}$$

(Remember that the inclusion of the trace/failure sets is in the opposite direction of the refinement.)

Question 2. Analyze the alternate system.

a. Show that the desired behavior refines the actual behavior:

$$\mathit{ASYS} \sqsubseteq_{\mathrm{F}} \mathit{DES}\,.$$

That is, every failure of the desired behavior is also a failure of the actual system. (Start by showing the corresponding trace-refinement.)

b. Show that the actual behavior *does not* refine the desired behavior:

$$\mathit{DES} \not\sqsubseteq_{\mathrm{T}} \mathit{ASYS}\,.$$

That is, the actual system has at least one undesirable trace.

c. Show that *ASYS* may actually *diverge*, in the sense that the system *without* the concealment of channels xmit and recv has arbitrarily long traces, that are all restricted down to exactly the same trace by the concealment. In other words, the system can perform an unbounded sequence of internal state transitions, without interacting with the environment (in this case, outputting anything on out).

General instructions As always, identify all the CSP laws you use, including section numbers. (When using 2.3.1 L7, it is enough to just refer to the law; you don't have to explicitly state the A, B, and C sets for each instance. Formally, for reasoning about conditionals ("$\triangleleft\triangleright$"), you would use laws 5.5.1 L7 and L8, but you may consider those to just repeat the definition of the construct.)

If you are not typesetting your answers (and possibly even if you are), please write all process parameters between parentheses: render XMIT_{n+1} as `XMIT(n+1)`, not `XMIT_n+1`, or similar. You may abbreviate process and channel names to their initial letters.