Solutions # 3

8.12 (a)⇒(b) According to the definitions of traces and fair traces, $fairtraces(A) \subseteq traces(A)$, therefore (b) holds.

(b)⇒(c) For any finite trace β that belongs to $traces(A)$, according to Theorem 8.7, there exists a fair trace $\beta' \in fairtraces(A)$ that starts with β, thus $\beta' \in traces(P)$ because of (b). By definition of Safety Property, $traces(P)$ is prefix-closed, which implies that $\beta \in traces(P)$, i.e. (c) holds.

(c)⇒(a) For any infinite trace β of A, we can have such a infinite sequence of traces $\beta_1, \beta_2, \ldots$ of A, where β_i is a prefix of β_{i+1} for any i, and β is the limit of this sequence. According to the limit-close property of $traces(P)$, $\beta \in traces(P)$, which means that (a) holds.

14.2 To the contrary, we suppose that there exists some trace β in $fairtraces(B) - fairtraces(A)$. Let $\beta = \alpha\pi\alpha'$. Without loss of generality, we assume that there exists some fair trace of A has the prefix α and for each $\beta' \in fairtraces(A)$ in form of $\alpha\pi'\alpha''$, it holds that $\pi \neq \pi'$, that is, as the common prefix of β and β', the length of α is 'maximal'.

Case.1 If $\pi = send_{i,j}(m)$, then according to the assumption, $\beta' = \alpha\pi'\alpha''$ is a fair trace of A for some π' and α'', where π' can never be $send_{i,j}(m)$. However, this is contra to the fact that A is a well defined I/O automaton, because $send_{i,j}(m)$ is an input action and as an I/O automaton, A should be input-enabled.

Case.2 If $\pi = receive_{i,j}(m)$, according to the semantics of B, for all those fair traces γ of B which starts with α, π should be the first receive action after α. Since $fairtraces(A) \subseteq fairtraces(B)$, this assertion also holds for fair traces of A, that is, for all those $\beta' = \alpha\pi'\alpha'' \in fairtraces(A)$, π' along with actions in α'' before $receive_{i,j}(m)$ have to all be send actions. However, send is an input action, which means A has no control over it and thus the above situation cannot be guaranteed, which leads to a contradiction.

In conclusion, $fairtraces(B) - fairtraces(A)$ is empty, thus plus $fairtraces(A) \subseteq fairtraces(B)$, we have $fairtraces(A) = fairtraces(B)$.

Remark: One thing worth to point out is that, in this problem, the only things that we can take as facts are:

(a) Both A and B are well-defined automata.

(b) B is a universal reliable FIFO channel.

(c) $ext(sig(A)) = ext(sig(B))$.

(d) $fairtraces(A) \subseteq fairtraces(B)$.

It is not correct to take it for granted that A is necessary to satisfy the FIFO axioms or have some FIFO semantics, although one can eventually deduce it from the above facts.

15.3 (a) ***AsynchHS*$_\mathrm{i}$ automaton**:

Signature:

Input:

$receive(v,c)_{i-1,i}$, v a UID, c an integer

$receive(v,c)_{i+1,i}$, v a UID, c an integer

Output:

$send(v,c)_{i,i+1}$, v a UID, c an integer

$send(v,c)_{i,i-1}$, v a UID, c an integer

$leader_i$

Internal:

$advance\text{-}phase_i$

States:

u, a UID, initially i's UID

$phase$, an integer, initially 0

$init_+$, $init_- \in \{0,1\}$, initially 1

$send_+, send_-$, FIFO queues, initially empty

$status \in \{unknown, chosen, reported\}$, initially $unknown$

Transitions:

$advance\text{-}phase_i$

Precondition:

$init_+ = init_- = 1$

Effect:

add $(u, 2^{phase})$ to $send_+$ and $send_-$

$init_+ := init_- := 0$

$phase := phase + 1$

$send(v,c)_{i,i+1}$

Precondition:

(v,c) is first on $send_+$

Effect:

remove first element from $send_+$

$send(v,c)_{i,i-1}$

Precondition:

(v,c) is first on $send_-$

Effect:

remove first element from $send_-$

$receive(v, c)_{i-1,i}$

Effect:

case

$c > 1$: if $v > u$ then add $(v, c-1)$ to $send_+$

if $v = u$ then $status := chosen$

$c = 1$: if $v > u$ then add $(v, 0)$ to $send_-$

if $v = u$ then $status := chosen$

$c < 1$: if $v \neq u$ then add (v, c) to $send_+$

if $v = u$ then $init_+ := 0$

endcase

$receive(v, c)_{i+1,i}$

Effect:

case

$c > 1$: if $v > u$ then add $(v, c-1)$ to $send_-$

if $v = u$ then $status := chosen$

$c = 1$: if $v > u$ then add $(v, 0)$ to $send_+$

if $v = u$ then $status := chosen$

$c < 1$: if $v \neq u$ then add (v, c) to $send_-$

if $v = u$ then $init_- := 0$

endcase

$leader_i$

Precondition:

$status = chosen$

Effect:

$status := reported$

Tasks:

$\{send(v, c)_{i,i+1}, send(v, c)_{i,i-1} : v \text{ a UID}, c \text{ an integer}\}$

$\{advance\text{-}phase_i\}$

$\{leader_i\}$

(b) Let $i_{\max}$ denote the index of the process with the maximum UID, and let $u$max denote its UID. It is sufficient to show:

i. No process other than $i_{\max}$ ever performs a *leader* output.

ii. Process $i_{\max}$ eventually performs a *leader* output.

By induction, we can easily show the following facts: for $c \geq 1$

i. if $i \neq i_{\max}$ and $j \in [i_{\max}, i)$, then (u_i, c) does not appear in $send_+$ of j;

ii. if $i \neq i_{\max}$ and $j \in (i, i_{\max}]$, then (u_i, c) does not appear in $send_-$ of j;

which directly imply that, if $i \neq i_{\max}$ and $c \geq 1$, then (u_i, c) never appear in either $send_+$ or $send_-$ of u_i. Therefore, we can make such statement that if $i \neq i_{\max}$ then $status_i = unknown$. Therefore no process other than $i_{\max}$ ever performs a *leader* output.

Again, by induction, we can show that, for every process, eventually $(u_{\max}, c)$ appears in both $send_+$ and $send_-$ for some $c \geq 1$, which guarantees the occurrence of

$receive(u_{\max}, c)_{i_{\max}-1, i_{\max}}$ and $receive(u_{\max}, c)_{i_{\max}+1, i_{\max}}$ for some $c \geq 1$. Therefore, process $i_{\max}$ eventually performs a *leader* output.

In conclusion, *AsynchHS* solves the leader-election problem.

(c) By the similar arguments as in Lemma 15.4, we have the upper bound on time complexity:

$$O((d+l)\sum_{i=1}^{\lceil \log n \rceil} 2^i) = O((d+l)n)$$

Remark: Some knowledge about queuing models may help.

(d) Instead of analyzing the time bound for process directly, we can think about the "life time" of messages. Note that the total number of steps that a UID might be delivered is up to

$$\sum_{i=1}^{\lceil \log n \rceil} 2^i = O(n)$$

and since the upper bound of arbitrary message delivery is d, the upper bound on the time complexity is $O(dn)$, which is also tight because we can easily construct an execution that simulates the synchronous case and approaches this upper bound.