Reactive modules

These exercises are mostly taken from Rajeev Alur and Thomas A. Henziger lectures on computer aided verification.

Exercise 1 — Synchronous circuits

Figure 1 defines a deterministic, synchronous, passive module for modeling logical Not.

module *SyncNot* **is**
  interface $out : \mathbb{B}$
  external $in : \mathbb{B}$
  atom controls out **awaits** in
    initupdate
      $[\!]$ $in' = 0 \rightarrow out' := 1$
      $[\!]$ $in' = 1 \rightarrow out' := 0$

Figure 1 – Module for logical Not

1. Propose a modelisation of logical And as a deterministic, synchronous, passive module, taking inspiration from the logical Not presented Figure 1.
2. Using de Morgan's law – $\neg(\neg x \wedge \neg y) = x \vee y$ – propose a modelisation of logical Or, using parallel composition of the modules for logical Not and logical And, variable renaming, and variable hiding.
3. What does the module SyncLatch of Figure 2 do ?
4. Represent the mysterious module of Figure 3 as a block diagram. Are you able to understand the behavior of this module ?
5. Use the module of Figure 3 to build a 3 bit counter.

module *SyncLatch* **is**
  private $state : \mathbb{B}$
  interface $out : \mathbb{B}$
  external $set, reset : \mathbb{B}$

  atom *ComputeOutput* **controls** *out* **reads** *state*
    init
      $[\!] \ true \rightarrow out' := \mathbb{B}$
    update
      $[\!] \ true \rightarrow out' := state$

  atom *ComputeNextState* **controls** *state* **awaits** *out*, *set*, *reset*
    initupdate
      $[\!] \ set' = 1 \rightarrow state' := 1$
      $[\!] \ reset' = 1 \rightarrow state' := 0$
      $[\!] \ set' = 0 \wedge reset' = 0 \rightarrow state' := out'$

FIGURE 2 – Module SyncLatch

module *Sync1BitCounter* **is**
  —**interface** *out*, *carry*
  —**external** *start*, *inc*
  hide *set*, *reset*, *z* **in**
    $\|$ $SyncLatch[set, reset, out]$
    $\|$ $SyncAnd[in_1, in_2, out := out, inc, carry]$
    $\|$ $SyncOr[in_1, in_2, out := carry, start, reset]$
    $\|$ $SyncNot[in, out := reset, z]$
    $\|$ $SyncAnd[in_1, in_2, out := inc, z, set]$

FIGURE 3 – Mysterious module

`Exercise 2` `Shared-variables protocols : mutual exclusion`

The mutual exclusion problem we consider is the following : there is a shared variable between two processes and one wants to ensure that this variable is only accessed by one process at a time. The part of each process accessing the shared variable is called *critical section*. The problem is then stated more clearly as follows : 1) *mutual exclusion* : no two processes have to be in their critical section at the same time, 2) *accessibility* : if some process requests an access to its critical section then it will have the opportunity to enter it at some time (as soon as the other process does not stay in its critical section forever).

1. Prove that SyncMutex (Figure 4) solves the mutual exclusion problem synchronously.

module Q_1 **is**
  interface $pc_1 : \{outC, reqC, inC\}$
  external $pc_2 : \{outC, reqC, inC\}$
  atom controls pc_1 **reads** pc_1, pc_2
    init
      $[\!]$ $true \rightarrow pc_1' := outC$
    update
      $[\!]$ $pc_1 = outC \rightarrow$
      $[\!]$ $pc_1 = outC \rightarrow pc_1' := reqC$
      $[\!]$ $pc_1 = reqC \wedge pc_2 \neq inC \rightarrow pc_1' := inC$
      $[\!]$ $pc_1 = inC \rightarrow$
      $[\!]$ $pc_1 = inC \rightarrow pc_1' := outC$

module Q_2 **is**
  interface $pc_2 : \{outC, reqC, inC\}$
  external $pc_1 : \{outC, reqC, inC\}$
  atom controls pc_2 **reads** pc_1, pc_2
    init
      $[\!]$ $true \rightarrow pc_2' := outC$
    update
      $[\!]$ $pc_2 = outC \rightarrow$
      $[\!]$ $pc_2 = outC \rightarrow pc_2' := reqC$
      $[\!]$ $pc_2 = reqC \wedge pc_1 = outC \rightarrow pc_2' := inC$
      $[\!]$ $pc_2 = inC \rightarrow$
      $[\!]$ $pc_2 = inC \rightarrow pc_2' := outC$

module *SyncMutex* **is** $Q_1 \parallel Q_2$

FIGURE 4 – Synchronous mutual exclusion

2. SyncMutex is active (why ?), modify it to make it passive (no variable needs to be added).

3. Prove that Pete (Figure 5) solves the mutual exclusion problem asynchronously.

module P_1 **is**
  interface $pc_1 : \{outC, reqC, inC\};\ x_1 : \mathbb{B}$
  external $pc_2 : \{outC, reqC, inC\};\ x_2 : \mathbb{B}$
  lazy atom controls pc_1, x_1 **reads** pc_1, pc_2, x_1, x_2
    init
      $[\!]\ true \to pc_1' := outC;\ x_1' := \mathbb{B}$
    update
      $[\!]\ pc_1 = outC \to pc_1' := reqC;\ x_1' := x_2$
      $[\!]\ pc_1 = reqC \wedge (pc_2 = outC \vee x_1 \neq x_2) \to pc_1' := inC$
      $[\!]\ pc_1 = inC \to pc_1' := outC$

module P_2 **is**
  interface $pc_2 : \{outC, reqC, inC\};\ x_2 : \mathbb{B}$
  external $pc_1 : \{outC, reqC, inC\};\ x_1 : \mathbb{B}$
  lazy atom controls pc_2, x_2 **reads** pc_1, pc_2, x_1, x_2
    init
      $[\!]\ true \to pc_2' := outC;\ x_2' := \mathbb{B}$
    update
      $[\!]\ pc_2 = outC \to pc_2' := reqC;\ x_2' := \neg x_1$
      $[\!]\ pc_2 = reqC \wedge (pc_1 = outC \vee x_1 = x_2) \to pc_2' := inC$
      $[\!]\ pc_2 = inC \to pc_2' := outC$

module $Pete$ **is hide** x_1, x_2 **in** $P_1 \parallel P_2$

FIGURE 5 – Asynchronous mutual exclusion using Peterson's protocol

4. Specify the three protocols mutual exclusion problem.

5. Propose a solution to the three protocols mutual exclusion problem, generalizing Peterson's protocol.

`Exercise 3` `Trajectories of compound modules`

The objective is to prove that for every pair P, Q of compatible modules, a sequence $\overline{s}$ of states in $\Sigma_{P\|Q}$ is an initialized trajectory of the compound module $P\|Q$ if and only if $\overline{s}[X_P]$ is an initialized trajectory of P and $\overline{s}[X_Q]$ is an initialized trajectory of Q.

1. Assume that the two modules have no private variables and that the interface variables of one are the external variables of the other. What can you deduce about the state spaces of P, Q, and $P\|Q$? Prove that $s \rightarrow_{P\|Q} t$ if and only if $s \rightarrow_P t$ and $s \rightarrow_Q t$. What can you say about the initial states of the compound module ?

2. Assume that the two modules have no variables in common. Remark that $\Sigma_{P\|Q} = \{s_1 \cup s_2 | s_1 \in \Sigma_P \wedge s_2 \in \Sigma_Q\}$. Prove that $(s_1 \cup s_2) \rightarrow_{P\|Q} (t_1 \cup t_2)$ if and only if $s_1 \rightarrow_P t_1$ and $s_2 \rightarrow_Q t_2$. What can you say about the initial states of the compound module ?

3. Consider the general case. Consider two states s and t of the compound module. Prove that $s \in \sigma^I_{P\|Q}$ if and only if $s[X_P] \in \sigma^I_P$ and $s[X_Q] \in \sigma^I_Q$. Prove that $s \rightarrow_{P\|Q} t$ if and only if $s[X_P] \rightarrow_P t[X_P]$ and $s[X_Q] \rightarrow_Q [X_Q]$.

4. Conclude.