# Exam Software Verification (TDDE34)

August 18, 2021

Examiner: ahmed.rezine@liu.se

- Time kl 08.00 - 12.00
- Submit a "main" pdf, word or text file. If you join pictures, reference them from the main file.
- This is an open book exam. You can access internet.
- It is however strictly forbidden to contact and discuss the exam, during the exam period, with any person other than the examiner, whether the person is related to the course or not.

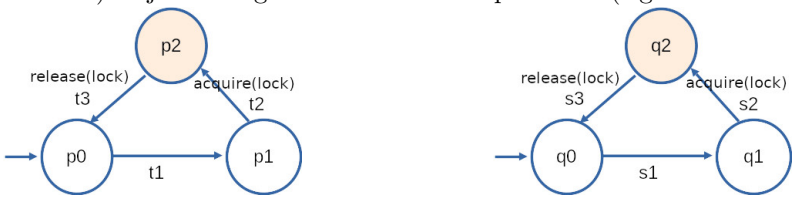
# 1 Branching time (6p)

Assume `Req` and `Ack` are atomic propositions. Express the following CTL properties using (boolean combinations of) **EG**, **EU** and the atomic propositions above:

- **AF**(`Ack`) (1p)
- **AG**(`!Req`) (1p)
- **AG**(**EF**(`Ack`)) (2p)
- **AG**(`Req`) $\Rightarrow$ **AG**(`Ack`) (2p)

# 2 Mutual exclusion (16p)

Assume the following description of a simple mutual exclusion algorithm for two processes **p** and **q**. State **p0** (resp. **q0**) is the initial state of process **p** (resp. process **q**). Process **p** (resp. process **q**) moves to state **p1** (resp. state **q1**) in case it wants to access its critical section. State **p2** (resp. **q2**) is the critical section of process **p** (resp. process **q**). Variable **lock** is shared by the two processes and can only be manipulated with the primitives **acquire** and **release**. It is initially free. When acquired, the lock becomes busy and cannot be acquired again until it is first released. Transitions are either lock acquisitions (e.g. **acquire(lock)** for transition **t2**) or lock releases (e.g. **release(lock)** for transition **s3**) or just change of state with no operations (e.g. transition **t1**).




## 2.1 Part A: (10p)

In the following, we use **@pi** to mean the proposition stating process **p** is at state **pi**. We do the same for process **q**. For instance, the proposition **@q2** is true in a configuration when process **q** is at its critical section. We use the following set of atomic propositions:

- Location propositions: $\{@\mathbf{pi} \mid 0 \leq \mathbf{i} \leq 2\} \cup \{@\mathbf{qi} \mid 0 \leq \mathbf{i} \leq 2\}$
- Values' propositions: $\{\mathbf{lock} = \mathbf{v} \mid \mathbf{v} \text{ in } \{\mathbf{free}, \mathbf{busy}\}\}$

Answer the following questions:

- The LTL formula **G**(!(@**p2** $\wedge$ @**q2**)) states that mutual exclusion is always respected. Does it hold? argue or give a run violating it (2p)

- Write an LTL formula corresponding to the starvation freedom of **p**, i.e., each time **p** wants to access its critical section it eventually succeeds. (2p)
- Write an LTL formula $\varphi_{p:alone}$ that states that it is always the case that if process **q** stays at **q0**, then each time **p** wants to access its critical section it eventually succeeds. (3p)
- Give a Büchi automaton for the formula $\varphi_{p:alone}$. Explain it. (3p)

### 2.2 Part B: (6p)

We assume the transitions are atomic. Transitions from different processes can be interleaved (a scheduler schedules one process at a time to execute a number of transitions). Transitions corresponding to lock acquisition are enabled if the corresponding process is at the start of the transition (e.g., **@q1** holds for **s2**) and the lock is free (i.e., **lock=free**). Transitions corresponding to lock release are enabled if the corresponding process is at the start of the transition (e.g., **@p2** holds for **t2**) and the lock is busy (i.e., **lock=busy**). Transitions that are not enabled cannot be fired. We write **En(t)** to mean transition **t** is enabled. We write **Ex(t)** to mean transition **t** is indeed executed. For instance **Ex(s2)** is true if **En(s2)** and process **q** moves from **q1** to **q2**. It is common to assume schedulers behave "reasonably". A way to account for this assumption is to restrict runs to those satisfying "reasonable" constraints. Consider the following constraint:

$\Phi$: for all transition **u** different from p1 of process **p** and different from **q1** of process **q**: GF(!**En(u)** or **Ex(u)**)

- Is restricting scheduler's behavior to $\Phi$ enough to ensure starvation freedom of process **p**? argue or give a counter-example. (3p)
- Is restricting scheduler's behavior to $\Phi$ enough to ensure $\varphi_{p:alone}$? argue or give a counter-example. (3p)

## 3 Symbolic representation (8p)

1. Assume integer variables $x_1, x_2$ and $x_3$. A pure function $f$ that associates integers to integers (assume the domain of $f$ contains all integers). An integer-indexed array $Arr$ containing integer values (assume the size of $Arr$ is infinite and all integers are valid indices). The following formula involves expressions in Linear Integer Arithmetic (LIA), Equality over Uninterpreted Functions (EUF) and Arrays (A) fragments.

$$\left(\begin{array}{c}(0 < x_1 \wedge x_1 < 4 \wedge 1 < x_2 \wedge x_2 < 3 \wedge x_1 + 1 = x_2) \\ \wedge \\ ((f(x_1) = f(1)) \Rightarrow (rd(wr(Arr, x_2, x_3), x_1) = (x_1 + x_3)))\end{array}\right)$$

   Give a model (i.e., values for the variables) for the formula if it is satisfiable, otherwise argue why it is not satisfiable. (4p)

2. Consider the formula $f(v_0, v_1, v_2, v_3)$ defined as $((v_0 \oplus v_1 \oplus v_2) = v_3)$ where $v_0, v_1, v_2$ and $v_3$ are boolean variables and $\oplus$ stands for the exclusive or binary operator. Give a BDD for $f$ assuming the order $v_0 < v_1 < v_2 < v_3$ (i.e., starting from the root, variable $v_0$ appears first, then variable $v_1$, ... etc). If it makes the submission simpler, you can draw the BDD on paper, take a picture and join it to your submission. (4p).

# 4 Partial and total correctness (10p)

Consider the following simple program:

$$
\begin{array}{l}
\{Q : x = 0 \wedge y = 0 \wedge i = n \wedge 0 \leq n\} \\
\textbf{do} \quad 0 < i \quad \rightarrow \quad i := i - 1; x := x + 3; y := y - 2 \\
\textbf{od} \\
\{R : x + y = n\}
\end{array}
$$

- Find a suitable invariant and use it to show that if the loop terminates after starting from a state satisfying $Q$ then it terminates in a state satisfying $R$ (6p)
- Find a suitable variant function and use it to show the loop terminates. (4p)