Name: _____

Student Number: _____

Signature: _____

THE UNIVERSITY OF NEW SOUTH WALES

COMP3153/9153
Algorithmic Verification

SAMPLE Final Exam

Term 1, 2020

Time Allowed: 2 Hours, plus 10 minutes reading time.

Total Number of Questions: 7

Answer **all** questions.

The questions **are not** of equal value.

You are permitted **two** hand-written, double-sided A4 sheets of notes.

Only write your answers on the provided booklets.

Your notes will be collected at the end of the exam.

Answers must be written in ink, with the exception of diagrams.

Drawing instruments or rules may be used.

Excessively verbose answers may lose marks.

There is a 3% penalty if your name and student number are not filled in correctly.

Papers may not be retained by students.

Question 1 (10 marks)

(a) (6 marks) Determine the truth value of the following statements, no justifications required. One mark is awarded for correct answers and *one mark is subtracted* for incorrect answers. No marks are awarded or subtracted when no answer is given.

i. The only property that is both safety and liveness is the trivial property $(2^{\mathcal{P}})^{\omega}$.

ii. In static analysis, a False Positive is a situation where an alarm is raised, even though there is no bug.

iii. Symbolic CTL model checking is based on the fix-point semantics of CTL.

iv. The CEGAR loop for predicate abstraction always terminates.

v. Two timed automata that are timed-abstract equivalent are also timed-equivalent.

vi. The diameter of an automaton is the greatest lower bound of the shortest distances between any connected states.

Solution:

i. True

ii. True

iii. True

iv. False

v. False

vi. False

(b) (4 marks) Answer the following questions with short answers (1-2 sentences):

i. Name one advantage and one disadvantage of the *abstraction refinement* method.

Solution: It produces a smaller state space, but only works for ACTL.

ii. Give a reason why "classical" LTL model checking using Büchi automata cannot be used for timed systems.

Solution: Büchi automata are closed under complementation, which is needed to compute the automata of the negated LTL formula. Timed Büchi automata are not closed under complementation, making this method unsuitable.

Question 2 (16 marks)

(a) (7 marks) Let p and q be atomic propositions. Consider the following two pairs of LTL/CTL-formulae. For each pair, determine whether they are equivalent. If your answer is 'Yes' provide a proof. If the answer is 'No' give a counterexample model.

i. $\mathbf{A}(\mathbf{A}\, p\, \mathbf{Until}\, q)\, \mathbf{Until}\, (\mathbf{A}\, q\, \mathbf{Until}\, p)$ and $\mathbf{A}\, p\, \mathbf{Until}\, q$;

Solution: No, the path $(p \wedge \neg q).(\neg p \wedge \neg q)^{\omega}$ (interpreted as tree) satisfies lhs, but not rhs.

ii. $(\mathbf{X}\, p)\, \mathbf{R}\, p$ and $p \wedge \mathbf{X}\, p$.

Solution: We first show $(\mathbf{X}\, p)\, \mathbf{Until}\, p \equiv p \vee (\mathbf{X}\, p)$.

$$
\begin{array}{rcl}
\underline{(\mathbf{X}\,p)\,\mathbf{Until}\,p} & \Leftrightarrow & p \vee (\mathbf{X}\,p \wedge \mathbf{X}\,((\mathbf{X}\,p)\,\mathbf{Until}\,p)) \\
& \Leftrightarrow & p \vee (\mathbf{X}\,p \wedge \mathbf{X}\,(p \vee (\mathbf{X}\,p \wedge \mathbf{X}\,((\mathbf{X}\,p)\,\mathbf{Until}\,p)))) \\
& \Leftrightarrow & p \vee (\mathbf{X}\,p \wedge ((\mathbf{X}\,p) \vee ((\mathbf{X}\,\mathbf{X}\,p) \wedge \mathbf{X}\,\mathbf{X}\,((\mathbf{X}\,p)\,\mathbf{Until}\,p)))) \\
& \Rightarrow & p \vee ((\mathbf{X}\,p) \wedge ((\mathbf{X}\,p) \vee (\mathbf{X}\,\mathbf{X}\,p))) \\
& \Leftrightarrow & p \vee ((\mathbf{X}\,p) \wedge (\mathbf{X}\,p)) \vee ((\mathbf{X}\,p) \wedge (\mathbf{X}\,\mathbf{X}\,p)) \\
& \Leftrightarrow & p \vee (\mathbf{X}\,p) \vee ((\mathbf{X}\,p) \wedge (\mathbf{X}\,\mathbf{X}\,p)) \\
& \Leftrightarrow & \underline{p \vee (\mathbf{X}\,p)} \\
& \Rightarrow & \underline{(\mathbf{X}\,p)\,\mathbf{Until}\,p}
\end{array}
$$

Using this result we now get

$$
\begin{array}{rcl}
(\mathbf{X}\,p)\,\mathbf{R}\,p & \equiv & \neg((\neg\mathbf{X}\,p)\,\mathbf{Until}\,(\neg p)) \\
& \equiv & \neg((\mathbf{X}\,\neg p)\,\mathbf{Until}\,(\neg p)) \\
& \equiv & \neg((\neg p) \vee (\mathbf{X}\,(\neg p))) \\
& \equiv & \neg((\neg p) \vee (\neg\mathbf{X}\,p)) \\
& \equiv & \neg\neg(p \wedge \mathbf{X}\,p) \\
& \equiv & p \wedge \mathbf{X}\,p
\end{array}
$$

(b) (9 marks) Let **AtNext** be a binary temporal operator. Informally, $\varphi\,\mathbf{AtNext}\,\psi$ means that "φ will be true the next time ψ is true, not assuming that φ or ψ will be true at all". Its formal semantics is given by

$$
\rho \models \varphi\,\mathbf{AtNext}\,\psi \;:\Longleftrightarrow\; \begin{cases} \big(\exists k \geq 1.\ (\rho[k] \models (\varphi \wedge \psi)\ \wedge\ \forall j.\ 1 \leq j < k \Rightarrow \rho[j] \models \neg\psi)\big) \vee \\ \neg(\exists k \geq 1.\ \rho[k] \models \psi) \end{cases}
$$

i. Prove that the next-operator **X** can be expressed using **AtNext**.

Solution: We prove $\mathbf{X}\,\varphi \equiv \varphi\,\mathbf{AtNext}\,\mathsf{true}$.

$$
\begin{array}{rl}
 & \rho \models \varphi\,\mathbf{AtNext}\,\mathsf{true} \\
\Leftrightarrow & \begin{cases} \big(\exists k \geq 1.\ (\rho[k] \models (\varphi \wedge \mathsf{true})\ \wedge\ \forall j.\ 1 \leq j < k \Rightarrow \rho[j] \models \mathsf{false})\big) \vee \\ \neg(\exists k \geq 1.\ \rho[k] \models \mathsf{true}) \end{cases} \\
\Leftrightarrow & \begin{cases} \big(\exists k \geq 1.\ (\rho[k] \models \varphi\ \wedge\ \forall j.\ 1 \leq j < k \Rightarrow \rho[j] \models \mathsf{false})\big) \vee \\ \mathsf{false} \end{cases} \\
\Leftrightarrow & \exists k \geq 1.\ (\rho[k] \models \varphi\ \wedge\ \forall j.\ 1 \leq j < k \Rightarrow \rho[j] \models \mathsf{false}) \\
\Leftrightarrow & \{\text{to satisfy second half, } k \leq 1\text{; hence } k = 1\} \\
 & \rho[1] \models \varphi\ \wedge\ \forall j.\ 1 \leq j < 1 \Rightarrow \rho[j] \models \mathsf{false} \\
\Leftrightarrow & \rho[1] \models \varphi \\
\Leftrightarrow & \rho \models \mathbf{X}\,\varphi
\end{array}
$$

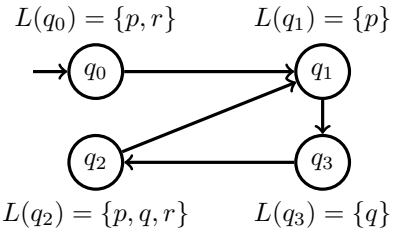
ii. Prove that the operator **AtNext** does not increase the expressiveness of LTL.

Solution: We prove $\varphi\,\mathbf{AtNext}\,\psi = (\mathbf{X}\,(\neg\psi\,\mathbf{Until}\,(\varphi \wedge \psi))) \vee (\mathbf{X}\,\mathbf{G}\,\neg\psi)$.

$$
\begin{aligned}
& \rho \models \varphi\,\mathbf{AtNext}\,\psi \\
\Leftrightarrow\ & \begin{cases} \exists k \geq 1.\ (\rho[k] \models (\varphi \wedge \psi)\ \wedge\ \forall j.\ 1 \leq j < k \Rightarrow \rho[j] \models \neg\psi) \text{ or} \\ \neg(\exists k \geq 1.\ \rho[k] \models \psi) \end{cases} \\
\Leftrightarrow\ & \begin{cases} \exists k \geq 1.\ (\rho[k] \models (\varphi \wedge \psi)\ \wedge\ \forall j.\ j < k-1 \Rightarrow \rho[j+1] \models \neg\psi) \text{ or} \\ \neg(\exists k \geq 1.\ \rho[k] \models \psi) \end{cases} \\
\Leftrightarrow\ & \begin{cases} \exists k \geq 0.\ (\rho[k+1] \models (\varphi \wedge \psi)\ \wedge\ \forall j.\ j < k \Rightarrow \rho[j+1] \models \neg\psi) \text{ or} \\ \forall k \geq 0.\ \rho[k+1] \models \neg\psi \end{cases} \\
\Leftrightarrow\ & \begin{cases} \rho[1] \models (\neg\psi\,\mathbf{Until}\,(\varphi \wedge \psi)) \text{ or} \\ \forall k \geq 0.\ \rho[k+1] \models \neg\psi \end{cases} \\
\Leftrightarrow\ & \rho \models (\mathbf{X}\,(\neg\psi\,\mathbf{Until}\,(\varphi \wedge \psi))) \text{ or} \\
& \rho \models (\mathbf{X}\,\mathbf{G}\,\neg\psi) \\
\Leftrightarrow\ & \rho \models (\mathbf{X}\,(\neg\psi\,\mathbf{Until}\,(\varphi \wedge \psi))) \vee (\mathbf{X}\,\mathbf{G}\,\neg\psi)
\end{aligned}
$$

Question 3 (18 marks)

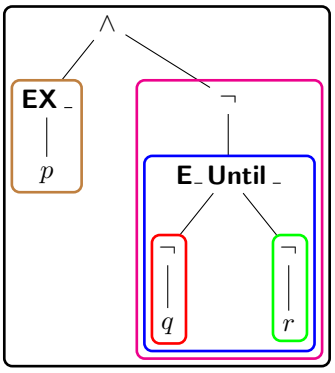
Here we have the automaton B:



For atomic propositions p, q, r let $\varphi = (\mathbf{EX}\,p) \wedge \neg(\mathbf{E}(\neg q)\,\mathbf{Until}\,(\neg r))$ be a CTL formula.

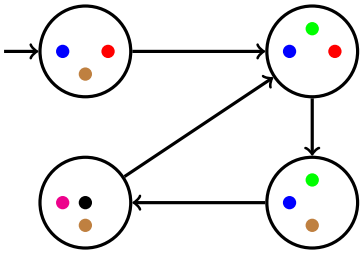
(a) Give the parse tree of the formula φ.

Solution:



(b) Manually run the CTL explicit-state marking algorithm on φ for Automaton B. Does $B \models \varphi$ hold? Explain your answer.

Solution:



Only the state q_2 satisfies the formula. Since it is not the initial state the automaton does not satisfy φ, i.e., $B \not\models \varphi$.

(c) Our marking algorithm only considers the temporal operators $\mathbf{EX}\,\varphi$, $\mathbf{E}\,\varphi\,\mathbf{Until}\,\psi$ and $\mathbf{A}\,\varphi\,\mathbf{Until}\,\psi$. To improve performance one introduces procedures for other temporal operators. Describe a procedure for "exists release", i.e. for $\mathbf{A}\,\varphi\,\mathbf{R}\,\psi$, which is defined as $\neg(\mathbf{E}(\neg\varphi)\,\mathbf{Until}\,(\neg\psi))$. (Pseudocode is not necessary)

Solution: We observe the following:

$$\begin{aligned}
\mathbf{A}\varphi\,\mathbf{R}\,\psi &= \neg(\mathbf{E}\neg\varphi\,\mathbf{Until}\,\neg\psi) \\
&= \neg(\neg\psi \vee (\neg\varphi \wedge \mathbf{EX}\,(\mathbf{E}\neg\varphi\,\mathbf{Until}\,\neg\psi))) \\
&= \psi \wedge (\varphi \vee \neg\mathbf{EX}\,(\mathbf{E}\neg\varphi\,\mathbf{Until}\,\neg\psi)) \\
&= \psi \wedge (\varphi \vee \mathbf{AX}\,\neg(\mathbf{E}\neg\varphi\,\mathbf{Until}\,\neg\psi)) \\
&= \psi \wedge (\varphi \vee \mathbf{AX}\,\mathbf{A}\varphi\,\mathbf{R}\,\psi) \\
&= (\psi \wedge \varphi) \vee (\psi \wedge \mathbf{AX}\,\mathbf{A}\varphi\,\mathbf{R}\,\psi)
\end{aligned}$$

This equation is closely related to the $\mathbf{A}\_\mathbf{Until}\_$-law $\mathbf{A}\varphi\,\mathbf{Until}\,\psi = \psi \vee (\varphi \wedge \mathbf{AX}\,(\mathbf{A}\varphi\,\mathbf{Until}\,\psi))$. That means that we can follow a similar algorithm as for $\mathbf{A}\_\mathbf{Until}\_$. However, the algorithm for $\mathbf{A}\_\mathbf{Until}\_$ used the fact that it was the smallest fixed point; here we need the largest – the can e.g. be seen when looking at the two possible paths $\mathbf{A}\varphi\,\mathbf{R}\,\psi$ allows:

- Either ψ remains true forever.
- φ and ψ becomes true at some point.

To accomplish this we can essentially run the $\mathbf{A}\_\mathbf{Until}\_$ algorithm backwards. We start by marking *all* states. Then, unmark all states where ψ is false. Then, unmark all states where φ is false that have an edge to those unmarked states, repeating until there are no more states to unmark.

Question 4 (10 marks)

Let φ and ψ be LTL formulae. Propose an algorithm to check if φ and ψ are equivalent. The following operations can be used without any explanations:

- *Build a Büchi automaton* A_ϕ that accepts the sequences satisfied by a given LTL-formula ϕ, e.g. by the construction using the local and eventuality automata.
- *Emptiness check* for Büchi automata, i.e. given a Büchi automaton B one can check if $L(B) = \varnothing$.
- *Build the cross product* of two Büchi automata, i.e. given two Büchi automata B_1, B_2, we can compute the product $B_1 \times B_2$.

All other steps used in the algorithm should be explained and justified.

Solution: We can construct Büchi automata for φ, ψ and their negations. We denote them by A_φ, A_ψ, $A_{\neg\varphi}$ and $A_{\neg\psi}$, resp. To check if $\varphi \Longleftrightarrow \psi$ we can check if the automata accept the same language, i.e., $L(A_\varphi) = L(A_\psi)$. Split the equation into two subsets and we are done. In detail, we have to check $L(A_\varphi) \subseteq L(A_\psi)$ and $L(A_\psi) \subseteq L(A_\varphi)$. Using the transformation given on several slides for LTL Model Checking, we calculate

$$\begin{array}{lll} & L(A_\varphi) \subseteq \overline{L(A_\psi)} & \wedge \quad L(A_\psi) \subseteq \overline{L(A_\varphi)} \\ \Leftrightarrow & L(A_\varphi) \cap \overline{L(A_\psi)} = \emptyset & \wedge \quad L(A_\psi) \cap \overline{L(A_\varphi)} = \emptyset \\ \Leftrightarrow & L(A_\varphi) \cap L(A_{\neg\psi}) = \emptyset & \wedge \quad L(A_\psi) \cap L(A_{\neg\varphi}) = \emptyset \\ \Leftrightarrow & L(A_\varphi \times A_{\neg\psi}) = \emptyset & \wedge \quad L(A_\psi \times A_{\neg\varphi}) = \emptyset \end{array}$$

The last line can be checked by running a standard LTL model checker (twice).

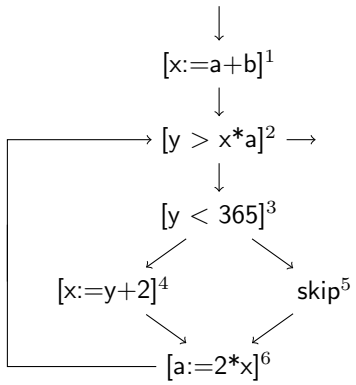
Question 5 (16 marks)

This exercise is about *Live Variables Analysis* of source code. Consider the following pseudo-code in a simple WHILE language:

```
[x:=a+b]1;
while [y > x*a]2 do (
  if [y<365]3
    then [x:=y+2]4;
    else skip5;
  [a:=2*x]6
);
```

(a) (2 marks) Give the control flow graph (CFG) for the above program.

Solution:



(b) (3 marks) Give the gen_{LV} and $kill_{\mathsf{LV}}$ function for each statement in the program.

Solution: Variables under consideration are x, y, a and b.

l	$\text{kill}_{\mathsf{LV}}(\ell)$	$\text{gen}_{\mathsf{LV}}(\ell)$
1	{x}	{a, b}
2	$\emptyset$	{a, x, y}
3	$\emptyset$	{y}
4	{x}	{y}
5	$\emptyset$	$\emptyset$
6	{a}	{x}

(c) (3 marks) Give the data flow equation for each node's entry (LV_{entry}) and each node's exit(s) (LV_{exit}).

Solution: LVA is a backwards *may analysis*.

$$\begin{aligned}
\mathsf{LV}_{\mathsf{entry}}(\ell) &= (\mathsf{LV}_{\mathsf{exit}}(\ell) \backslash \mathrm{kill}_{\mathsf{LV}}(\ell)) \cup \mathrm{gen}_{\mathsf{LV}}(\ell) \\
\mathsf{LV}_{\mathsf{exit}}(\ell) &= \begin{cases} \emptyset & \text{if } \ell \in \mathit{final} \\ \bigcup\{\mathsf{LV}_{\mathsf{entry}}(\ell') \mid (\ell', \ell) \in \mathit{flow}^R\} & \text{otherwise} \end{cases}
\end{aligned}$$

So, for each node, we have:

$$\begin{array}{ll}
\mathsf{LV}_{\mathsf{exit}}(1) = \mathsf{LV}_{\mathsf{entry}}(2) & \mathsf{LV}_{\mathsf{entry}}(1) = (\mathsf{LV}_{\mathsf{exit}}(1) \backslash \{\mathsf{x}\}) \cup \{\mathsf{a}, \mathsf{b}\} \\
\mathsf{LV}_{\mathsf{exit}}(2) = \emptyset \cup \mathsf{LV}_{\mathsf{entry}}(3) & \mathsf{LV}_{\mathsf{entry}}(2) = \mathsf{LV}_{\mathsf{exit}}(2) \cup \{\mathsf{a}, \mathsf{x}, \mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(3) = \mathsf{LV}_{\mathsf{entry}}(4) \cup \mathsf{LV}_{\mathsf{entry}}(5) & \mathsf{LV}_{\mathsf{entry}}(3) = \mathsf{LV}_{\mathsf{exit}}(3) \cup \{\mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(4) = \mathsf{LV}_{\mathsf{entry}}(6) & \mathsf{LV}_{\mathsf{entry}}(4) = (\mathsf{LV}_{\mathsf{exit}}(4) \backslash \{\mathsf{x}\}) \cup \{\mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(5) = \mathsf{LV}_{\mathsf{entry}}(6) & \mathsf{LV}_{\mathsf{entry}}(5) = \mathsf{LV}_{\mathsf{exit}}(5) \\
\mathsf{LV}_{\mathsf{exit}}(6) = \mathsf{LV}_{\mathsf{entry}}(2) & \mathsf{LV}_{\mathsf{entry}}(6) = (\mathsf{LV}_{\mathsf{exit}}(6) \backslash \{\mathsf{a}\}) \cup \{\mathsf{x}\}
\end{array}$$

(d) (8 marks) Compute the least fix point of the equation set, i.e., the result of the live variables analysis by giving the resulting LV_{entry}, LV_{exit} for all nodes after resolving the equation set.

Solution:
Let's calculate the fixpoint. We start with $\mathsf{LV}_{\mathsf{entry}}(i) = \emptyset$.
(A chaotic characterisation is quicker as we e.g. see that y is needed but never set).

$$\begin{array}{ll}
\mathsf{LV}_{\mathsf{exit}}(1) = \emptyset & \mathsf{LV}_{\mathsf{entry}}(1) = \{\mathsf{a}, \mathsf{b}\} \\
\mathsf{LV}_{\mathsf{exit}}(2) = \emptyset & \mathsf{LV}_{\mathsf{entry}}(2) = \{\mathsf{a}, \mathsf{x}, \mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(3) = \emptyset & \mathsf{LV}_{\mathsf{entry}}(3) = \{\mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(4) = \emptyset & \mathsf{LV}_{\mathsf{entry}}(4) = \{\mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(5) = \emptyset & \mathsf{LV}_{\mathsf{entry}}(5) = \emptyset \\
\mathsf{LV}_{\mathsf{exit}}(6) = \emptyset & \mathsf{LV}_{\mathsf{entry}}(6) = \{\mathsf{x}\}
\end{array}$$

Iteration 1 (rhs starts indicating new bits – of course only the changes need to be considered):

$$\begin{array}{ll}
\mathsf{LV}_{\mathsf{exit}}(1) = \{\mathsf{a}, \mathsf{x}, \mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(1) = \{\mathsf{a}, \mathsf{b}, \underline{\mathsf{y}}\} \\
\mathsf{LV}_{\mathsf{exit}}(2) = \{\mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(2) = \{\mathsf{a}, \mathsf{x}, \mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(3) = \{\mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(3) = \{\mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(4) = \{\mathsf{x}\} & \mathsf{LV}_{\mathsf{entry}}(4) = \{\mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(5) = \{\mathsf{x}\} & \mathsf{LV}_{\mathsf{entry}}(5) = \{\underline{\mathsf{x}}\} \\
\mathsf{LV}_{\mathsf{exit}}(6) = \{\mathsf{a}, \mathsf{x}, \mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(6) = \{\mathsf{x}, \underline{\mathsf{y}}\}
\end{array}$$

Iteration 2:

$$\begin{array}{ll}
\mathsf{LV}_{\mathsf{exit}}(1) = \{\mathsf{a}, \mathsf{x}, \mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(1) = \{\mathsf{a}, \mathsf{b}, \mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(2) = \{\mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(2) = \{\mathsf{a}, \mathsf{x}, \mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(3) = \{\underline{\mathsf{x}}, \mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(3) = \{\underline{\mathsf{x}}, \mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(4) = \{\mathsf{x}, \underline{\mathsf{y}}\} & \mathsf{LV}_{\mathsf{entry}}(4) = \{\mathsf{y}\} \\
\mathsf{LV}_{\mathsf{exit}}(5) = \{\mathsf{x}, \underline{\mathsf{y}}\} & \mathsf{LV}_{\mathsf{entry}}(5) = \{\mathsf{x}, \underline{\mathsf{y}}\} \\
\mathsf{LV}_{\mathsf{exit}}(6) = \{\mathsf{a}, \mathsf{x}, \mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(6) = \{\mathsf{x}, \mathsf{y}\}
\end{array}$$

Iteration 3:

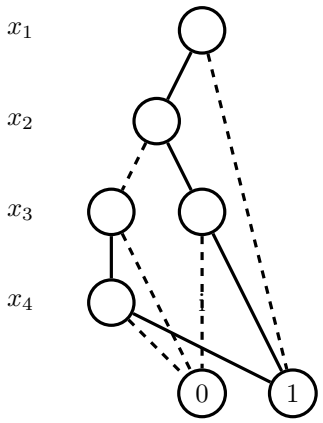
$$\begin{array}{ll} \mathsf{LV}_{\mathsf{exit}}(1) = \{\mathsf{a},\mathsf{x},\mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(1) = \{\mathsf{a},\mathsf{b},\mathsf{y}\} \\ \mathsf{LV}_{\mathsf{exit}}(2) = \{\underline{\mathsf{x}},\mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(2) = \{\mathsf{a},\mathsf{x},\mathsf{y}\} \\ \mathsf{LV}_{\mathsf{exit}}(3) = \{\mathsf{x},\mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(3) = \{\mathsf{x},\mathsf{y}\} \\ \mathsf{LV}_{\mathsf{exit}}(4) = \{\mathsf{x},\mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(4) = \{\mathsf{y}\} \\ \mathsf{LV}_{\mathsf{exit}}(5) = \{\mathsf{x},\mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(5) = \{\mathsf{x},\mathsf{y}\} \\ \mathsf{LV}_{\mathsf{exit}}(6) = \{\mathsf{a},\mathsf{x},\mathsf{y}\} & \mathsf{LV}_{\mathsf{entry}}(6) = \{\mathsf{x},\mathsf{y}\} \end{array}$$

Fix point reached.

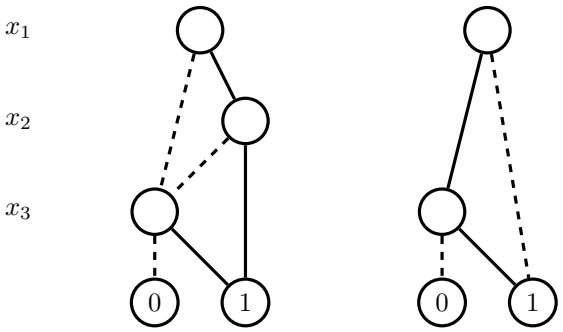
Question 6 (15 marks)

(a) (5 marks) Give the reduced binary decision diagram (ORBDD) for $x_1 \Rightarrow (x_3 \wedge (x_2 \vee x_4))$, using the order of the variables $x_1 < x_2 < x_3 < x_4$, i.e., x_1 is at the top of the tree.

Solution:

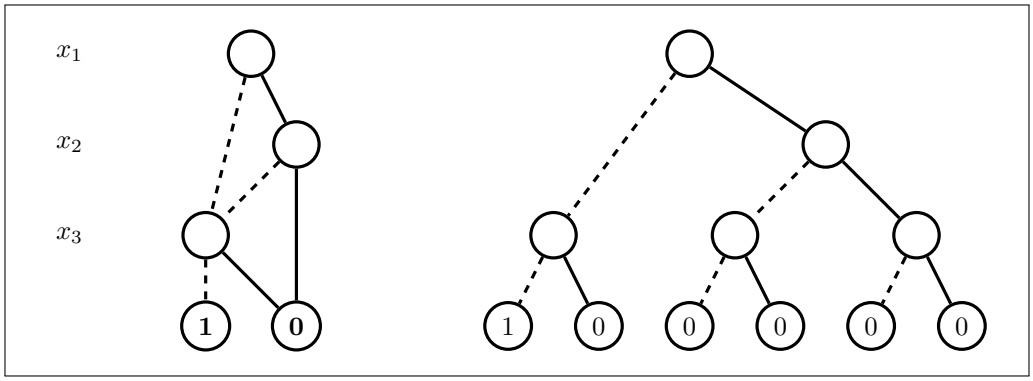


(b) (6 marks) Given the following two ORBDDs, encoding the formulas φ and ψ



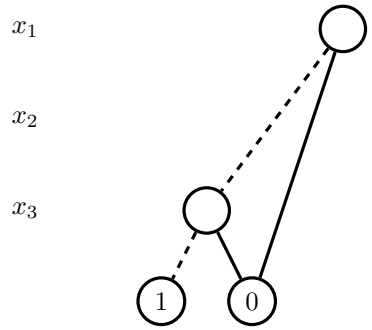
Here a solid line indicates the Boolean value true, and a dashed line the value false.
Determine a BDD for $\neg\varphi \wedge \psi$, using among others the operator $\otimes$ of the lecture.
(The diagram does not need to be a tree, nor need it be reduced.)

Solution: Left hand side is $\neg\varphi$; right hand side is the answer.



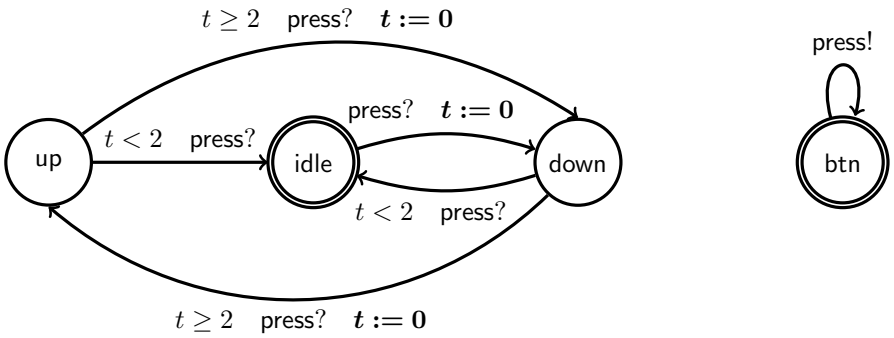
(c) (4 marks) Reduce the BDD for part (b).

Solution: As the entire right-hand side maps to 0 we can reduce this bit entirely



Question 7 (15 marks)

Here we have two timed automata, Elevator and Button:



Locations are labelled with their names. Double-circled locations are *initial* states. Each transition is labelled with a guard (if present), a communication channel in sans-serif font (if present), and finally an assignment in **boldface** (if present).

The two automata synchronise via the binary handshake channel press. They must synchronise. The model contains only one clock t. The cross-product of the timed automata is denoted by $S = \text{Elevator} \times \text{Button}$.

The model is intended to describe an elevator controlled by one button only. There are three states: the elevator can go up, down or stand still (idle).

(a) (3 marks) Describe the behaviour of the timed automaton Elevator in English words.

Solution: After the first press, the elevator goes down. If it is followed by a second press in less than 2 time units (t.u.), it stops. Otherwise after 2 t.u. the press will make the elevator go in the other direction up. Same applies from up: if a second press is issued in less than 2 t.u. after the previous one, the elevator stops. Otherwise a second press after 2 t.u. changes direction.

(b) (6 marks) The atomic propositions up, down and idle represent the corresponding locations of the automaton Elevator in the product S. Assume the following TCTL specifications:

$$\begin{aligned}\varphi_1 &= \mathbf{AG}\ \big(\mathsf{down} \Rightarrow (\mathbf{AF}_{\geq 0}\ \mathsf{idle})\big)\\ \varphi_2 &= \mathbf{AG}\ \big(\mathsf{up} \Rightarrow (\mathbf{EF}_{\leq 1}\ \mathsf{idle})\big)\\ \varphi_3 &= \mathbf{AG}\ \big(\mathsf{up} \Rightarrow (\mathbf{EF}_{<2}\ \mathsf{down})\big)\end{aligned}$$

For each $\varphi_i, 1 \leq i \leq 3$, does S satisfy φ_i? Justify your answer.

Solution: φ_1 not satisfied: from $(\mathsf{idle}, 0)$ go to $(\mathsf{down}, 0)$ and wait for ever.
φ_2 satisfied: from $(\mathsf{up}, t = \delta)$ if $\delta < 2$ we can go to idle directly. Otherwise if $\delta \geq 2$, we press twice within 1 time unit and reach idle.
φ_3 satisfied: from $(\mathsf{up}, t = \delta)$ if $\delta \geq 2$ we can go to down directly. Otherwise if $\delta < 2$, we press twice within 2 time unit and reach down.

(c) (6 marks) We want to enforce a press! action to occur infinitely often.
The specification for this is $P(k)$: the first press! action has to occur within $(\leq)$ k time units after the system started, and any press! must be followed by another press! action within $(\leq)$ k time units, for a fixed integer $k \geq 0$.

i. What do you need to add to Button to enforce this behaviour?

ii. Does the product Elevator $\times$ Button(k) contain deadlocks, where Button(k) is the modified Button automaton?

Solution: We need to add a new clock y and an invariant in Button. Invariant is $[y \leq k]$ and the press! transition must reset y.
There are no deadlocks as press? is enabled in any state of Elevator and thus cannot block the product.

END OF EXAM