1. Consider the Kripke structure $K$ depicted below. The states are $\{s_0, s_1, s_2\}$, with $s_0$ as the initial state. We use $Prop = \{stable, waiting, x=y\}$. Which propositions hold (and otherwise) at each state can be seen below.

Consider the property $\phi = \lozenge\square(x=y)$.

(a) What does the formula say?

**Answer:** Eventually, $x = y$ will continue to hold.

(b) What is its negation?

**Answer:** Well, $\neg\square\phi = \lozenge\neg\phi$ and $\neg\lozenge\phi = \square\neg\phi$.

So, the negation of the above is: $\square\lozenge(x \neq y)$.

(c) Give a Buchi automaton $A_\neg$, that represent this negation.

**Answer:** A standard Buchi with $b_1$ as the accepting state:

(d) Construct the automaton $K \cap A_\neg$.

**Answer:** We’ll first convert the Kripke $K$ to its Buchi equivalent. We’ll move the labels from the states to the arrows. You need to do it in such a way, that the set of generated sentences are still the same. You can do this in two ways:

i. We notice that when an execution pass a state $s$ in $K$, its next step must pass an outgoing arrow from $s$. So, we move labels from any state $s$ to all its outgoing arrows.

An issue arises if you have a terminal state $t$ (a state with no successor). However, recall that in our context we have assumed that our state automata do not contain such a state.

This approach gives us the Buchi automaton shown below; we have a single accepting set $F$ consisting of all states in the automaton:
ii. Analogously, you can move labels of $s$ to all its incoming arrows. For each initial state $s_0$ of $K$, we additionally add a new dummy initial state $z_0$, and an arrow from $z_0$ to $s_0$, so that we can also move the labels of $s_0$ to an arrow.

This gives the following automaton: we have a single accepting set $F$ consisting of all states in the automaton:

I’ll take the first version, since it is a bit smaller. Below $K$ refers to this Buchi-equivalent of the original $K$.

Let $\Sigma_K = \{ s_0, s_1, s_2 \}$ be $K$’s set of states. Similarly let $\Sigma_{A\neg} = \{ b_0, b_1 \}$ be $A\neg$’s set of states.

Let’s call $K \cap A\neg$, the automaton $I$. The states of $I$ will be drawn from $\Sigma_K \times \Sigma_{A\neg}$.

$I$ would start from $(s_0, b_0)$; that is, the combined starting states of both $K$ and $A\neg$. $I$ will contains only transitions that would be allowed by both $K$ and $A\neg$. So, there is a transition :

$$(s, b) \xrightarrow{A} (t, c)$$

only if $s \xrightarrow{A} t$ and $b \xrightarrow{A} c$.

Do keep in mind that the notation like $b \xrightarrow{p \notin A} c$ that we used in the picture of $A\neg$ represents a family of arrows from $b$ to $c$, each is labelled by a subset $A$ of $Prop$ such that $p \notin A$.

The accepting states of $I$ would be all those states $(s, b)$ where $s$ is accepting in $K$ and $b$ is accepting in $A\neg$. However, since all states of $K$ is accepting, only $b$ determines if $(s, b)$ would be accepting.

Now we can quite easily construct $I$:
Where $A = \{ \text{stable}, x=y \}$ and $B = \{ \text{waiting}, x=y \}$.

The states $\{ (s_0, b_1), (s_2, b_1) \}$ above are accepting.

(e) So, does $K$ satisfies the property $\phi$?

**Answer:** No. For example run $(s_0, b_0), (s_1, b_0), (s_0, b_1), (s_1, b_0), (s_0, b_1), ...$ is an accepting run in $I$. And therefore $L(I)$ cannot be empty. Notice that this run is also a counter example for the property $\Diamond \Box (x = y)$.

That is, if you project the run to the states of $K$:

$s_0, s_1, s_0, s_1, ...$

it shows you a run in the original program that violates $\Diamond \Box (x = y)$.

2. Verify if following properties are valid properties of $K$ from No. 1:

(a) $\Box \Box (x = y)$

(b) $\neg \text{waiting} \cup (\text{waiting} \land x=y)$

3. What does this formula $\phi = \Box (\text{waiting} \rightarrow (\text{waiting} \land \text{stable}))$ say? Verify if it is a valid property of $K$.

**Answer:**

It says: whenever $\text{waiting}$ holds, then either it holds forever, or at some point it stops to hold; but then $\text{stable}$ has to hold.

Let’s do some calculation to simplify $\neg \phi$:

$\neg (\Box (\text{waiting} \rightarrow (\text{waiting} \land \text{stable})))$

$= \Diamond (\text{waiting} \land \neg (\text{waiting} \land \text{stable}))$

$= \Diamond (\text{waiting} \land (\text{waiting} \land \neg \text{stable}) \cup (\neg \text{waiting} \land \neg \text{stable}))$

This formula has a nested $\cup (\Diamond$ actually abbreviates $true \cup g$), which makes it a bit harder to figure out what the corresponding Buchi automaton. Here is the automaton:

```
   waiting∈ , stable∉       waiting, stable ∉
   *  b1                      *  b2

   waiting∈ , stable∉
   *  b2
```

with a single accepting state, namely $b_2$.

We use $∈$ and $∉$ abbreviation on the label. To remind you, their meaning are as follows (see also the LN):
• $s \stackrel{p \in}{\rightarrow} t$ represents a family of arrows $s \stackrel{A}{\rightarrow} t$ such that $p \in A$.
• $s \stackrel{p,q \in}{\rightarrow} t$ represents a family of arrows $s \stackrel{A}{\rightarrow} t$ such that $p \in A$ AND $q \in A$.
• $s \stackrel{p,q \notin}{\rightarrow} t$ represents a family of arrows $s \stackrel{A}{\rightarrow} t$ such that $p \notin A$ AND $q \notin A$.
• $s \stackrel{l_1,l_2}{\rightarrow} t$ abbreviates two families of arrows: $s \stackrel{l_1}{\rightarrow} t$ and $s \stackrel{l_2}{\rightarrow} t$.

The set $F$ of the accepting states of this intersection automaton consists of all $(s, b_2)$, for any $s \in \Sigma_K$. However, no such state is reachable from the initial state, as you can see above. Therefore, the language of the above automaton is empty. This implies that no counter example exists for $\phi$. Therefore $\phi$ is valid on $K$. 

\[
\begin{array}{ccc}
(s_0, b_0) & \xrightarrow{\{\text{stable, } x=y\}} & (s_0, b_1) \\
\downarrow & & \downarrow \\
(s_1, b_0) & \xrightarrow{\{\text{waiting, } x=y\}} & (s_2, b_0) \\
\downarrow & & \downarrow \\
\emptyset & \xrightarrow{\{\text{waiting, } x=y\}} & \emptyset \\
\end{array}
\]