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 $Prop = \{ \text{stable, waiting, } x=y \}$. Which propositions hold (and otherwise) at each state can be seen below.

\[
\begin{align*}
&s_0 : \{ \text{stable, } x=y \} \\
&s_1 : \emptyset \\
&s_2 : \{ \text{waiting, } x=y \}
\end{align*}
\]

Consider the property $\phi = \Diamond \Box (x=y)$.

(a) What does the formula say?

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

(b) What is its negation?

**Answer:** $\Box \Diamond (x \neq y)$

(c) Give a Buchi automaton $A_-$ that represent this negation.

**Answer:**

\[
\begin{tikzpicture}
  \node (b0) at (0,0) {$b_0$};
  \node (b1) at (1,0) {$b_1$};
  \draw[->] (b0) -- node[above]{$x=y$} (b1);
  \draw[->, bend left] (b0) to node[below]{$*\$} (b1);
  \draw[->, bend right] (b0) to node[below]{$*$} (b1);
\end{tikzpicture}
\]

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

**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 \( \Sigma_\neg = \{ b_0, b_1 \} \) be \( \neg A \)’s set of states.

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

\( I \) would start from \( (s_0, b_0) \); that is, the combined starting states of both \( K \) and \( \neg A \). \( I \) will contains only transitions that would be allowed by both \( K \) and \( \neg A \). 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 \( b \not\in A \) that we used in the picture of \( \neg A \) represents a family of arrows from \( b \) to \( c \), each is labelled by a subset \( A \) of \( \text{Prop} \) such that \( p \not\in A \).

The accepting states of \( I \) would be all those states \( (s, b) \) where \( s \) is accepting in \( K \) and \( b \) is accepting in \( \neg A \). 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 \( (s_0, b_0), (s_1, b_0), (s_0, b_0), (s_1, b_0), (s_0, b_1), ... \) 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 \( K \):

\[
\begin{align*}
& (s_0, b_0), (s_0, b_1), \ldots
\end{align*}
\]

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) \( \Box \Diamond (x = y) \)

(b) \( \neg \text{waiting} \boxdot (\text{waiting} \wedge x = y) \)

3. What does this formula \( \phi = \Box (\text{waiting} \rightarrow (\text{waiting} \mathcal{W} \text{stable})) \) 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 \( \neg \phi \):

\[
\begin{align*}
\neg(\Box (\text{waiting} \rightarrow (\text{waiting} \mathcal{W} \text{stable}))) &= \\
\Diamond (\text{waiting} \wedge \neg (\text{waiting} \mathcal{W} \text{stable})) &= \\
\Diamond (\text{waiting} \wedge ((\text{waiting} \wedge \neg \text{stable}) \mathcal{U} (\neg \text{waiting} \wedge \neg \text{stable})))
\end{align*}
\]

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

We have two accepting groups. For the \( \Diamond \)-part, we have \( F_1 = \{ b_1, b_2 \} \). For the \( \mathcal{U} \) -part, we have \( F_2 = \{ b_2 \} \). However, this can be simplified. It is sufficient to require a single accepting state, namely \( b_2 \).
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 $K$. 

\[ \{ \text{stable, } x=y \} \]

\[ \{ \text{waiting, } x=y \} \]

\[ \{ \text{wait, } x=y \} \]

\[ \{ \text{waiting, } x=y \} \]

\[ \{ \text{waiting, } x=y \} \]