Some notes on LTL formulas

The until-connectives

We have seen two versions of the until-connective: *until* ($\mathcal{U}$) and *unless* or *weak until* ($\mathcal{W}$). The difference is that in the case of $p\,\mathcal{U}\,q$, we actually enforce q, whereas we do not in the case of $p\,\mathcal{W}\,q$.
One application for the untils, we can see below: to describe a sequence where something happens, and then something else. When using the until, there is an equivalence that might be of use:

$$(p_1 \wedge \neg p_2)\,\mathcal{U}\,p_2 \\ \Leftrightarrow \\ p_1\,\mathcal{U}\,p_2$$

This might seem counter intuitive at first, the first formula seems to express more than the second. But if we think of what it means, it makes sense: the part $\neg p_2$ in the first part is implicit in the second. Why? Because as soon as we observe a state in the computation sequence that satisfies p_2, we are done. Consequently, we need not explicitly enforce it in the prefix. The same thing is true for $\mathcal{W}$.
Thanks Martin Stigge for pointing this out.

2.3, part c

We want to find an LTL formula for "The first position in which q is true must coincide with or be preceded by a p- position.".
We did this in the lecture, but I misread the sentence, thinking it said "immediately preceded"instead of preceded". Now we'll do it properly.
Let's think about what this means. First of all I'd claim that this does in no way enforce that we'll reach a state that satisfies q. The claim is a disguised if-statement. In general, statements like "When p...", "Whenever p...", "The n:th p-state..." and so on, allows there to be no p-state in a computation.
Thinking more of what the property we want to specify says, we realise that it is a statement about a prefix of the computation. Another way of saying it would be "p has to be true before, or at the latest at the same time as q is true". This naturally seems to be two cases, so let's treat them separately, and we start with the latter because it is simpler.
Since we are working with a prefix of the computation, we know that the until-connective is the way to go. The prefix we want, is that $(\neg p \wedge \neg q)$ is true for some (possibly empty) sequence of states, then there is a state with $(p \wedge q)$ true, and then we no longer care. Again, since we do not require there to actually be a true q, we formulate this property as $(\neg p \wedge \neg q)\,\mathcal{W}\,(p \wedge q)$.
For the former property, we want a sequence where first $(\neg p \wedge \neg q)$ is true for some (possibly empty) sequence of states, then there should be a state where $(p \wedge \neg q)$ is true, then some interleaving of $(p \wedge \neg q)$ and $(\neg p \wedge \neg q)$ and finally a state satisfying $(\neg p \wedge q)$ or $(p \wedge q)$. To encode this seems horrible, until we

realise that we are being too precise. For example, to say that we should satisfy $(\neg p \wedge q)$ or $(p \wedge q)$ is the same as saying that we should satisfy just q. Also, we can see that since we do not require there to actually be a q-state, this last condition is not needed. And since the last condition is not needed, we do not need the sequence leading up to it, i.e., we do not need to enforce the "interleaving of $(p \wedge \neg q)$ and $(\neg p \wedge \neg q)$" either. This means that we only need to write down a formula expressing that "first $(\neg p \wedge \neg q)$ is true for some (possibly empty) sequence of states, then there should be a state where $(p \wedge \neg q)$ is true". This again looks like a perfect job for the weak until, and so we formulate the property as $(\neg p \wedge \neg q)\ \mathcal{W}\ (p \wedge \neg q)$.
Putting this together with a disjunction, we get

$$((\neg p \wedge \neg q)\ \mathcal{W}\ (p \wedge \neg q)) \vee ((\neg p \wedge \neg q)\ \mathcal{W}\ (p \wedge q))$$

Being extra bright, we can recognize that there is a striking similarity between the two sides of the disjunction. The only thing that differs is that there is a $\neg q$ in the latter part of the first where there is a q in the latter part of the latter. This calls for simplification, and it is fairly easy to see that we can write the whole thing as just $(\neg p \wedge \neg q)\ \mathcal{W}\ p$.
At this point, we do do some matching against the equivalence noted above. We see that

$$\begin{array}{c} (\neg p \wedge \neg q)\ \mathcal{W}\ p \\ \Leftrightarrow \\ (\neg q \wedge \neg p)\ \mathcal{W}\ p \\ \Leftrightarrow \\ \neg q\ \mathcal{W}\ p \end{array}$$

Interpreting this and thinking hard about what it actually means, we conclude that we are correct: the formula can be read as "first $\neg q$ for some time, and then p becomes true (and possibly q as well, we don't know)". We can also read it just as it stands: "not q unless p", or with more words: "we shall not see q, unless we have seen p".
Note that we are obviously not required to take this detour through the big formula and then minimize. We can just write it down from the start. But that requires that we are able to formulate it in words in the right way from the start.

2.2, part a

Let's formulate the property "The processes A and B alternate in accessing the file".
As before, and this is generally the case, it might greatly help if we reformulate the property. One candidate reformulation would be "If A is in the critical section, then the next time a process enters the critical section, it should be B, and vice versa". With this reformulation, we uncover the conditional structure of this property.

First, lets define some terms as we did in the irl session. We define "ac" as "A is in the critical section", and "bc" equivalently. As already noted, we have an implication to start with, so the property looks like "$ac \implies p$" for some p.
The property p we can read out above: "The next time a process enters the critical section, it should be B". Let's look closer at what this means. It means that there is some sequence of ac-states, followed by a (possibly empty) sequence of $\neg ac \wedge \neg bc$ states, followed in turn by a sequence of bc states. As discussed in the last problem, we use the until-connective to do this. We start, as in the last problem from the end to formulate the property. Saying that there should be "a (possibly empty) sequence of $\neg ac \wedge \neg bc$ states, followed in turn by a sequence of bc states", is easily expressed as $(\neg ac \wedge \neg bc)\ \mathcal{W}\ bc \Leftarrow \neg ac\ \mathcal{W}\ bc$. This should be preceded by a non empty sequence of ac states, so we just continue without thinking: $ac\ \mathcal{W}\ (\neg ac\ \mathcal{W}\ bc)$. The whole property then, is

$$ac \implies (ac\ \mathcal{W}\ (\neg ac\ \mathcal{W}\ bc))$$

We add the symmetric case, and get the whole bonanza:

$$\begin{array}{c} ac \implies (ac\ \mathcal{W}\ (\neg ac\ \mathcal{W}\ bc)) \\ \wedge \\ bc \implies (bc\ \mathcal{W}\ (\neg bc\ \mathcal{W}\ ac)) \end{array}$$

Now there is the point that neither the weak until, not the normal one, enforces the c_1 in $c_1\ \mathcal{W}\ c_2$. But in this case it the implication acts to enforce the condition at least once, so it is not a problem.
Another point is that sequence $(ac \wedge bc), (\neg ac \wedge \neg bc), (ac \wedge bc), (\neg ac \wedge \neg bc), \ldots,$ actually fulfills this property. This is rather counter intuitive, since we might want alternation to mean that at most one process has the resource. If we want this, we can just add it to the property with a conjunct. This is an easy exercise. Do it! It also might be worth thinking about whether there is a computation in this particular example that satisfies such a sequence.
There are yet other comments to this formula. For example, it does not enforce that there is ever any process that enters the critical section. It also does not enforce that once any process grabs the resource, it ever releases it. And what would happen if we substitute $\mathcal{W}$ for $\mathcal{U}$?