# LTL Concepts Rubric

Use the tags below to assess incorrect student answers to the LTL survey.

The tags focus on high-level problems: concepts and misconceptions.

For each answer, follow the below rubric process. Explain your reasoning in a comment. Likewise, if you wish to provide additional commentary about certainty beyond the tag (e.g., uncertain if the student made a typo vs. misconception), do so in a **comment**.

---

Step 1: Is this answer correct up to missing parentheses? (e.g., the student may have made a typo-level mistake omitting some parentheses) If yes, apply **Precedence** and **STOP.**

> *Rationale: We don't want to guess about removing and/or rearranging parentheses, because that opens a huge space of possibilities. We don't want to keep coding after a Precedence (hence STOP) because it's hard to get agreement after; the Precedence is a fork, and coders might take different paths. We didn't want to force the coder to use intuition to judge between "misconception" and a typo, hence this unified code. (e.g., "always {Red in Panel.lit implies (after Red not in Panel.lit and after Red in Panel.lit)}" ANON wasn't sure if it was a typo or a BadStateIndex misconception.*
>
> *Wrong-parens example: eventually{Red in Panel.lit } and eventually{Red not in Panel.lit} and always{ { Red in Panel.lit and after{Red not in Panel.lit}} => after{always{Red not in Panel.lit}} }*

Step 2: Is this answer correct for one of our "reasonable variants"? If so, tag the problem-appropriate **Miscomm label** and continue to code **relative to the variant instead of the "correct answer"**. Go back and check **Precedence** for the new variant.

- **B.1** = "Red light blinks on/off forever"
  Instead of "If Red turns on, then blinks forever"
- **D.1** = "Red turns on in the 1st or 2nd step"
  Instead of "turns on eventually, not necessarily 1st step"
- **M.1** = "Red cannot stay lit 4 steps in a row"
  Instead of 3 steps
- **V.1** = "Blue light must flip from OFF to ON"
  Instead of merely being on

- **Y.1** = “Red is eventually lit & then stays lit for a finite # steps”
  Instead of starting lit
- **Y.2** = “Red must be on initially”
  A learner might add a constraint that Red starts lit. Our staff solution allows Red to start unlit (if it never becomes lit) because of how it uses Until.

The meaning of a label like “B.1” is: *this is the 1st misconception for the question from the B column in the sheet*

> *Rationale: We often found that student answers appeared to be based on ambiguities in our English prompts. We identified a small number of "reasonable" mis-reads, which correspond to the Miscomm labels. But still, many of those answers involved mistakes, which we wanted to quantify and examine. It's possible that the coders will disagree about what the student was trying to target, and we have provided individual tags in order to detect this.*

Step 3: Apply semantic tags. Use the rubric below. If a formula is very complex / confusing, and/or if there are several potential interpretations about what went wrong, don’t apply any tags.

> *Rationale: Want to identify and exclude cases where something is so off that the coder was unsure if it was problem A or problem B etc. We can examine it later for a discussion (shows the extent of the confusion you can get from LTL ... imagine doing synthesis based off this formula, how far off is that result from the original English spec?!)*

- **BadProp** : Mis-used/swapped a logical operator or atom (may have misunderstood the definition of "and" or our English description of a light being on or off).
  - Expected `a0 => a1` but learner wrote `a0 and a1`
  - Expected `a0 => a1` but learner wrote `a0 => a0`

- **BadStateIndex** : Wrote a correct term or subterm that applies at the wrong time/state index. This code should not be applied in cases where fan-out quantification (F, G, U) is omitted, included erroneously, or uses the wrong quantifier, but only when the time index at which a subterm is evaluated is incorrect (either singly, or multiply in the scope of a temporal quantifier).
  - Expected `a0 U (a1 and F(a2))` but learner wrote `(a0 U a1) and F(a2)`
  - Expected `a0 and X(a1)` but learner wrote `a0 and a1`
  - Expected `X(a0)` but learner wrote `XX(a0)`
  - Expect `G(a0 => X(a1))` but learner wrote `G(a0 => a1)`.

> *Rationale: excluding quantifier errors here, to avoid overlap with **ImplicitF**, **ImplicitG, and BadStateQuantification**.*

- **BadStateQuantification** : Mis-used/swapped a temporal quantifier (G, F, U).

- Expected `F(a0)` but learner wrote `G(a0)` (wrong polarity)
- Expected `a0 U a1` but learner wrote `G(a0) U a1` (quantifier where shouldn't have)

**Note:** if the answer appears to use "after" (X) as a binary operator (e.g., expected `a0 => X(a1)` but learner wrote `a0 X a1`) record this in a comment.

> *Rationale: this code is about mis-use of a fan-out quantifier, NOT about mis-use of X, since that is about a concrete index when doing the recursive descent, not exists/forall. Hence, we pulled out the binary-after misconception we noticed before to avoid tainting the semantics of the code.*

- **ImplicitF** : Assumed that a formula (or subformula) must happen at some point. This might be as simple as a missing F quantifier, but might also be the apparent belief that some subterm is forced to happen eventually.
  - Expected `F(a0)` but learner wrote `a0`
  - Expected `F(a0) and G(a0 => a1)` but learner wrote `G(a0 => a1)`

- **ImplicitG** : Assumed that a formula (or subformula) constrains all future states, rather than just the current state index.
  - Expected `G(a0)` but learner wrote `a0`
  - Expected `G(a0 => G(a1)) but learner wrote `G(a0 => a1)`

- **OtherImplicit** : Omitted some constraints from a formula, perhaps assuming that traces will have some baseline behavior if unconstrained (this might be "the light is off unless I turn it on", or "the light is on unless I turn it off" or "things stay as they are", etc. ) **Do not** apply this tag when **ImplicitF** or **ImplicitG** would work.
  - Expected `!a0 U a0` but learner wrote `F(a0)`
  - Expected `a0 U G(!a0)` but learner wrote `a0 and F(G(!a0))`
  - Expected `F(a0) and G(a0 => XG(!a0))` but learner wrote `F(a0 and XG(!a0))`

> *Rationale: We noticed that there were some answers that underconstrained the problem but did not fall into ImplicitG/ImplicitF (such as saying "eventually a0" and never actually forcing a0 to remain off until that point). Some, but not all, of these could possibly be captured by an ImplicitU code, but that seemed confusing since the second subterm would be implicit and unstated.*

- **WeakUntil** : Confused U with weak-U.
  - Expected `a0 U a1`

> *Rationale: This code is meant to capture confusion about the fact that `until` forces the eventual truth of its 2nd argument. It is possible there is some overlap with Miscomm*

*tags here: if students can interpret our problem sentences as admitting vacuous truth...) We might not need this tag for this part of the quiz; we made this tag in response to the traces section.*