COSI135: Fall 2017
Final Project Option: CTL Branching Future

This option is a modification of the assignment for problem set 5, with the inclusion of some computational tree logic (CTL) operators to allow for generation of models that include branching in the future.

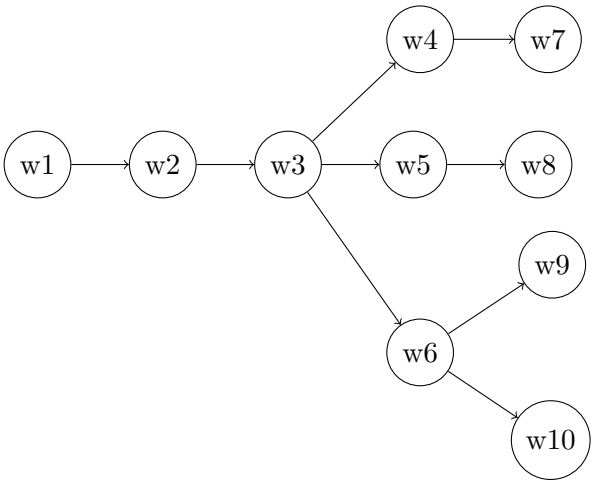
In problem set 5, you dealt with validity and satisfiability in a linear temporal logic (LTL) model. You are to create a structure that allows for branching, and then implement `isValid`, `isSatisfied` and `isSatisfiable` functions in this module in the manner of PS5.

You should start with the starter code for PS5. If you are satisfied with your solution for PS5, you can begin there, too. Most edits should be made in `World.hs`, and will consist mostly of creating the structure to allow for branching worlds and incorporating the CTL operators into your validity and satisfiability functions.

1. **Branching structure**

   You should add some more worlds (a total of about twelve should do) and populate them with propositions from your parser. You will also need to change the structure of your model. In the existing `World.hs`, the model is represented by a list of worlds, which is adequate for representing the linear structure. For a branching structure, you will need to rearchitect this slightly. A simple way to represent branching is as a list of tuples, where each tuple is a pair representing two worlds that have an accessibility relation between them, going *from* the first member of the pair *to* the second. For example:

   `[(w1, w2), (w2, w3), (w3, w4), (w3, w5), (w3, w6), (w4, w7), (w5, w8), (w6, w9), (w6, w10)]` represents a tree structure that looks like this:



2. **CTL operators**

   You should make a slight change to the `TProp` type: the `prop` field should become a [`Prop`] instead of a simple `Prop` for reasons that will be explained below.

   The `TemporalOperator` type is now more properly a "CTLOperator," and thus `TProp` is now a "CTLProp" (you can rename them if you like, or you can just remember that you're dealing with a slightly different logical construct than you were in PS5. You will need to add 4 CTL operators to this type: E, A, X and W. You can remove the past operators P and H as they will not be used. The modified TemporalOperator type may look something like this.

   Definitions of the CTL operators follow:

   - E — There **Exists** at least one path starting from the current world. $E\phi$ means there exists at least one path starting from the current world along which ϕ is true at some point.
   - A — **All** paths starting from the current world. $A\phi$ means ϕ is true at some point along all paths starting from the current world.
   - X — **Next**: ϕ has to hold at the next state.
   - W — A given proposition holds (**Weakly**) Until a different proposition becomes true. ϕ W ψ means ϕ is true along a given path until ψ becomes true. W is sometimes read as "unless." Because this operator takes two propositions instead of one, it was necessary to change the `prop` field of `TProp` to a list.
   - F — A given proposition **Finally** (eventually) holds somewhere along the given path. $F\phi$ means ϕ is true somewhere along the subsequent path. This definition of F has changed from the LTL definition.
   - G — A given proposition holds **Globally** along the given path. $G\phi$ means ϕ is true at all points along the subsequent path. This definition of G has changed from the LTL definition.

E and A are operators over all paths, while X, W, F, and G are path-specific operators. This means they can be combined in certain ways. In strict CTL (as opposed to CTL*), you may combine one all-path operator with one path-specific operator, which leads to certain interpretations. For instance, $EG\phi$ means that there exists one path starting from the current world where ϕ is true at all states along it. $AF\phi$ means that ϕ is eventually true along all paths starting from the current world. $A[\phi\ W\ \psi]$ means that along all paths starting from the current world, ϕ holds until ψ is true. The Wikipedia page on CTL provides some more information: `http://en.wikipedia.org/wiki/Computation_tree_logic`

With this understanding, the first aim of this task is to implement `isValid`, `isSatisfied` and `isSatisfiable` functions for this new branching model and these new CTL operators, using the same interpretations of validity and satisfiability as you used in PS5. A valid CTL proposition holds at all worlds in your model. A satisfiable CTL proposition holds at at least one world in the model. As before, you should populate the worlds in your model with some propositions, and all these propositions should be in the present tense. Your validity and satisfiability functions should be able to handle all valid combinations of CTL operators (one all-path operator and one path-specific operator) and all tenses that are available in your parser (use the same parser you built for PS4 and used for PS5, but you are not required to make any changes to it). (85% of credit)

The second aim is to make your CTL generative (15%). You can choose to do one of the following options:

- Extend it with a set of logical operators: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$. You are not required to have a fully functioning first order logic model checking, but your TProp should be changed to handle $EG.(p \Rightarrow q)$ and $EG.p \Rightarrow AF.q$ where p and q are your atomic propositions, and your `isValid`, `isSatisfied` and `isSatisfiable` functions should be able to work with the modification.
- Extend it for one more level of nested temporal operators, for example, the following expression $AF.EG.p$ means "It's always possible that at some point p will hold for the rest of time"

Your submission should also contain a write-up describing how you attacked this problem, what avenues you explored, and what problems you ran into. There is no length requirement, but you should include some use cases for testing and you should describe your strategy, process, and assumptions in some depth.