Applied Formal Methods
Homework 2: Temporal Logic

Prof. Kristin Yvonne Rozier

1. Advantages of Temporal Logic

In order to be able to analyze specifications of a system (aka descriptions of the behaviors the system should have) we have to encode them in a way that enables us to reason about them formally.

Once we have system specifications in temporal logic we can use them in all sorts of different kinds of analysis, from the design stage to the testing stage to the runtime stage of our system! For example, we might use model checking in the design stage to verify partial designs while we are creating them. Then during the verification stage we might use a code-level verifier to check our generated software; we might also use a modification of the model checking technology to generate test cases for system testing of both hardware and software. We can use runtime monitoring and runtime system health management methods during system execution time to analyze its behavior in real time and add an additional layer of assurance that we are adhering to our original specifications. The great thing is, in many cases the same set of temporal logic specifications are useful throughout all of these stages of system design and operation!

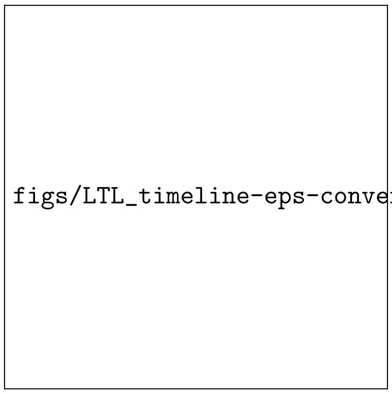
1.1. History of Temporal Logics

Continuous systems necessarily involve a notion of time. Propositional logic is not expressive enough to describe such real systems. Yet, English descriptions are even less precise once we involve time. Consider, for example, the following English sentences:

- I said I would see you on Tuesday.
- "This is the worst disaster in California since I was elected." [California Governor Pat Brown]
- "When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone." [Kansas State Legislature, early 1890's]

As a result, Amir Pnueli introduced the notion of using temporal logics, which were originally developed by philosophers for investigating how time is used in natural language arguments, to reason about concurrent systems [**?**]. (Burstall [**?**] and Kröger [**?**] independently proposed the use of weaker forms of temporal reasoning about computer programs at roughly the same time.) Temporal logics are modal logics geared toward the description of the temporal ordering of events. For most safety-critical systems, tying the system model to an explicit universal clock is overkill; it exacerbates the state explosion problem without adding value to the verification process. It is sufficient simply to guarantee that events happen in a certain partial (or, in some cases, total) order. Temporal logics were invented for this purpose. Because they describe the ordering of events in time without introducing time explicitly, temporal logics are particularly effective for describing concurrent systems [?]. Temporal logics conceptualize time in one of two ways. Linear temporal logics consider every moment in time as having a unique possible future. Essentially, they reason over a classical timeline. In branching temporal logics, each moment in time may split into several possible futures. In essence, these logics view the structure of time as a tree, rooted at the current time, with any number of branching paths from each node of the tree.

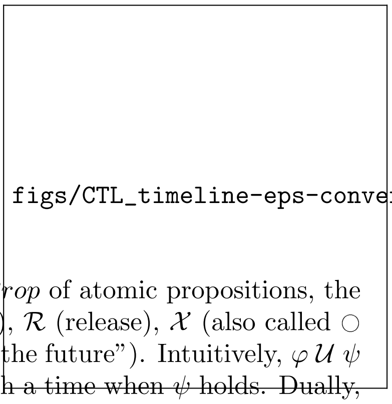
Pnueli's historic 1977 paper [?] proposed the logic LTL. This logic extended propositional logic with temporal operators $\mathcal{G}$ (for "globally") and $\mathcal{F}$ (for "in the future"), which we also call $\Box$ and $\Diamond$, respectively, and introduced the concept of *fairness*, which ensures an infinite-paths semantics. LTL was subsequently extended to include the $\mathcal{U}$ ("until") operator originally introduced by Kamp in 1968 [?] and the $\mathcal{X}$ ("next time") operator from the temporal logic $\mathscr{UB}$ (the *unified* system of *branching* time) [?]. In 1981, Clarke and Emerson extended $\mathscr{UB}$, thereby inventing the branching temporal logic they named Computational Tree Logic (CTL) and, with it, CTL model checking. Lichtenstein and Pnueli defined model checking for LTL in 1985 [?]. (For a more detailed history of the technical developments which lead from Prior's early observations on time and Church's logical specification of sequential circuits to LTL and automata-theoretic model checking, see [?].) The combination of LTL and CTL, called CTL*, was defined by Emerson and Halpern in 1986, though there are currently no industrial model checkers for this language.[1]



(a) Linear Temporal Logic

2. Linear Temporal Logic

Linear Temporal Logic (LTL) reasons over linear traces through time. At each time instant, there is only one real future timeline that will occur. Traditionally, that timeline is defined as starting "now," in the current time step, and progressing infinitely into the future.



(b) Branching Temporal Logic

Figure 1: Two visions of the structure of time.

Linear Temporal Logic (LTL) formulas are composed of a finite set *Prop* of atomic propositions, the Boolean connectives $\neg$, $\wedge$, $\vee$, and $\rightarrow$, and the temporal connectives $\mathcal{U}$ (until), $\mathcal{R}$ (release), $\mathcal{X}$ (also called $\bigcirc$ for "next time"), $\Box$ (also called $\mathcal{G}$ for "globally") and $\Diamond$ (also called $\mathcal{F}$ for "in the future"). Intuitively, $\varphi\,\mathcal{U}\,\psi$ states that either ψ is true now or φ is true now and φ remains true until such a time when ψ holds. Dually, $\varphi\;\mathcal{R}\;\psi$, stated φ *releases* ψ, signifies that ψ must be true now and remain true until such a time when φ is true, thus releasing ψ. $\mathcal{X}\varphi$ means that φ is true in the next time step after the current one. Finally, $\Box\varphi$ commits φ to being true in every time step while $\Diamond\varphi$ designates that φ must either be true now or at some future time step. We define LTL formulas inductively:

Definition 1. *For every $p \in Prop$, p is a formula. If φ and ψ are formulas, then so are:*

[1]There was one research prototype CTL* model checker called AltMC (**Alt**ernating Automata-based **M**odel **C**hecker). Visser and Barringer created AltMC in 1998 and explained how it could be integrated into the industrial model checker SPIN [?].

$\neg\varphi$	$\varphi \wedge \psi$	$\varphi \rightarrow \psi$	$\varphi\,\mathcal{U}\,\psi$	$\Box\varphi$
	$\varphi \vee \psi$	$\mathcal{X}\varphi$	$\varphi\,\mathcal{R}\,\psi$	$\Diamond\varphi$

Furthermore, we define the closure of LTL formula φ, $cl(\varphi)$, as the set of all of the subformulas of φ and their negations (with redundancies, such as φ and $\neg\neg\varphi$, consolidated). LTL formulas describe the behavior of the variables in *Prop* over a linear series of time steps starting at time zero and extending infinitely into the future.[2] We satisfy such formulas over *computations*, which are functions that assign truth values to the elements of *Prop* at each time instant [**?**]. In essence, a computation path π satisfies a temporal formula φ if φ is true in the zeroth time step of π, π_0.

Definition 2. *We interpret LTL formulas over computations of the form $\pi : \omega \rightarrow 2^{Prop}$, where ω is used in the standard way to denote the set of non-negative integers. We also use* iff *to abbreviate "if and only if." We define $\pi, i \vDash \varphi$ (computation π at time instant $i \in \omega$ satisfies LTL formula φ) as follows:*

- $\pi, i \vDash p$ *for* $p \in Prop$ *iff* $p \in \pi(i)$.
- $\pi, i \vDash \neg\varphi$ *iff* $\pi, i \nvDash \varphi$.
- $\pi, i \vDash \varphi \wedge \psi$ *iff* $\pi, i \vDash \varphi$ *and* $\pi, i \vDash \psi$.
- $\pi, i \vDash \varphi \vee \psi$ *iff* $\pi, i \vDash \varphi$ *or* $\pi, i \vDash \psi$.
- $\pi, i \vDash \mathcal{X}\varphi$ *iff* $\pi, i+1 \vDash \varphi$.
- $\pi, i \vDash \varphi\,\mathcal{U}\,\psi$ *iff* $\exists j \geq i$, *such that* $\pi, j \vDash \psi$ *and* $\forall k, i \leq k < j$, *we have* $\pi, k \vDash \varphi$.
- $\pi, i \vDash \varphi\,\mathcal{R}\,\psi$ *iff* $\forall j \geq i$, *if* $\pi, j \nvDash \psi$, *then* $\exists k$, $i \leq k < j$, *such that* $\pi, k \vDash \varphi$.
- $\pi, i \vDash \Box\varphi$ *iff* $\forall j \geq i$, $\pi, j \vDash \varphi$.
- $\pi, i \vDash \Diamond\varphi$ *iff* $\exists j \geq i$, *such that* $\pi, j \vDash \varphi$.

We take $\models (\varphi)$ to be the set of computations that satisfy φ at time 0, i.e., $\{\pi : \pi, 0 \vDash \varphi\}$. We define the prefix *of an infinite computation π to be the finite sequence starting from the zeroth time step, $\pi_0, \pi_1, \ldots, \pi_i$ for some $i \geq 0$.*

Equivalences. While we have presented the most common LTL syntax, operator equivalences allow us to reason about LTL using a reduced set of operators. In particular, the most common minimum set of LTL operators is $\neg$, $\vee$, $\mathcal{X}$, and $\mathcal{U}$. From propositional logic, we know that $(\varphi \rightarrow \psi)$ is equivalent to $(\neg\varphi \vee \psi)$ by definition and that $(\varphi \wedge \psi)$ is equivalent to $\neg(\neg\varphi \vee \neg\psi)$ by DeMorgan's law. We can define $(\Diamond\varphi)$ as $(true\,\mathcal{U}\varphi)$. (Similarly, $(\Box\varphi) \equiv (false\,\mathcal{R}\,\varphi)$.) The *expansion laws* state that $(\varphi\,\mathcal{U}\,\psi) = \psi \vee [\varphi \wedge \mathcal{X}(\varphi\,\mathcal{U}\,\psi)]$ and $(\varphi\,\mathcal{R}\,\psi) = \psi \wedge [\varphi \vee \mathcal{X}(\varphi\,\mathcal{R}\,\psi)]$. The operators $\Box$ and $\Diamond$ are logical duals as $(\Box\varphi)$ is equivalent to $(\neg\Diamond\neg\varphi)$ and $(\Diamond\varphi)$ is equivalent to $(\neg\Box\neg\varphi)$. Finally, $\mathcal{U}$ and $\mathcal{R}$ are also logical duals. Since this last relationship is not intuitive, we offer proof below. (Incidentally, $\mathcal{X}$ is the dual of itself: $(\neg\mathcal{X}\varphi) \equiv (\mathcal{X}\neg\varphi)$.)

[2]Though we will not discuss them here, some variations of LTL also consider time steps that have happened in the past. For example, we could add past-time versions of $\mathcal{X}$ and $\mathcal{U}$ called $\mathcal{Y}$ (also called $\ominus$ for "previous time" or "yesterday") and $\mathcal{S}$ (since), respectively. Past-time LTL was first introduced by Kamp in 1968 [**?**] but does not add expressive power to the future-time LTL defined here [**?**].

Examples of LTL properties:

- *Liveness: "Every request is followed by a grant"*
  $\Box(request \rightarrow \Diamond grant)$
- *Invariance: "At some point, p will hold forever"*
  $\Diamond\Box p$
- *"p oscillates every time step"*
  $\Box((p \wedge \mathcal{X}\neg p) \vee (\neg p \wedge \mathcal{X}p))$
- *Safety: "p never happens"*
  $\Box\neg p$
- *Fairness: "p happens infinitely often"*
  $(\Box\Diamond p) \rightarrow \varphi$
- *Mutual exclusion: "Two processes cannot enter their critical sections at the same time"*
  $\Box\neg(in\_CS_1 \wedge in\_CS_2)$
- *Partial correctness: "If p is true initially, then q will be true when the task is completed"*
  $p \rightarrow \Box(done \rightarrow q)$

Informally, $\varphi\ \mathcal{U}\ \psi$ signifies that either ψ is true now (in the current time step) or φ is true now and for every following time step until such a time when ψ is true. Note that this operator is also referred to as *strong until* because ψ *must* be true at some time. Conversely, *weak until*, sometimes included as the operator $\mathcal{W}$, is also satisfied if φ is true continuously but ψ is never true. Formally,

$$\pi, i \vDash \varphi\ \mathcal{W}\ \psi \text{ iff either } \forall j \geq i, \pi, j \vDash \varphi \text{ or } \exists j \geq i, \text{ such that } \pi, j \vDash \psi \text{ and } \forall k, i \leq k < j, \text{ we have } \pi, k \vDash \varphi.$$

$\mathcal{R}$ is the dual of $\mathcal{U}$, so $\varphi\ \mathcal{R}\ \psi = \neg(\neg\varphi\ \mathcal{U}\ \neg\psi)$. We say that "$\varphi$ releases ψ" because ψ must be true now and, in the future, $\neg\psi$ implies φ was previously true. Note that at that point in the past, both φ and ψ were true at the same time and that φ may never be true, as long as ψ is always true. We can define $\mathcal{R}$ in terms of the $\mathcal{U}$-operator using the following lemma, where iff is used in the standard way to abbreviate "if and only if."

Lemma 1. *$\neg(\neg a\ \mathcal{U}\ \neg b) = a\ \mathcal{R}\ b$.*

Proof. We use the formal semantic definitions of $\mathcal{U}$ and $\mathcal{R}$, given in [**?**].

$$\begin{array}{rll}
\pi, i \models \xi\ \mathcal{U}\ \psi & \text{iff} & \text{for some } j \geq i, \text{ we have } \pi, j \models \psi \\
 & & \text{and for all } k,\ i \leq k < j, \text{ we have } \pi, k \models \xi. \\
\pi, i \models \neg(\xi)\ \mathcal{U}\ \neg(\psi) & \text{iff} & \text{for some } j \geq i, \text{ we have } \pi, j \models \neg(\psi) \\
 & & \text{and for all } k,\ i \leq k < j, \text{ we have } \pi, k \models \neg(\xi). \\
\pi, i \models \neg(\xi)\ \mathcal{U}\ \neg(\psi) & \text{iff} & ((\exists\ j \geq i :\ \pi, j \models \neg(\psi))
\end{array}$$

$$
\begin{array}{rll}
 & & \wedge(\forall\ k,\ i \le k < j :\ \pi, k \models \neg(\xi))). \\
\pi, i \models \neg(\neg(\xi)\ \mathcal{U}\ \neg(\psi)) & \text{iff} & \neg((\exists\ j \ge i :\ \pi, j \models \neg(\psi)) \\
 & & \wedge(\forall\ k,\ i \le k < j :\ \pi, k \models \neg(\xi))). \\
 & \text{iff} & (\neg(\exists\ j \ge i :\ \pi, j \models \neg(\psi)) \\
 & & \vee\neg(\forall\ k,\ i \le k < j :\ \pi, k \models \neg(\xi))). \\
 & \text{iff} & ((\forall\ j \ge i :\ \pi, j \not\models \neg(\psi)) \\
 & & \vee(\exists\ k,\ i \le k < j :\ \pi, k \not\models \neg(\xi))). \\
 & \text{iff} & (\forall\ j \ge i :\ \pi, j \models \psi \\
 & & \vee\exists\ k,\ i \le k < j :\ \pi, k \models \xi). \\
 & \text{iff} & (\forall\ j \ge i :\ \pi, j \not\models \psi \\
 & & \rightarrow \exists\ k,\ i \le k < j :\ \pi, k \models \xi). \\
 & \text{iff} & \text{for all } j \ge i \text{ if } \pi, j \not\models \psi, \\
 & & \text{then for some } k,\ i \le k < j \text{ we have } \pi, k \models \xi. \\
 & \text{iff} & \pi, i \models \xi\ \mathcal{R}\ \psi.
\end{array}
$$

3. Visualizing LTL[3]

Another way to think of computation is as infinite traces of sensor signals. For example, you might be in the cockpit of an aircraft watching a display the values of signals for p and q that might look something like this:

[3]Some of the exercises and graphics in this section of the homework are borrowed with permission from Ian Barland, John Greiner, and Moshe Vardi.

figs/always_if_p_q_timeline.png

Figure 2: A timing diagram for a computation π with propositions p and q. Each propositional variable's value during the trace is depicted as a line. When the line is high, the proposition is true; when the line is low, it is false.

In this example, p and q are propositions from $Prop$. Since in the compuation π we have p being true at (for example) index 1, we write $\pi, 1 \vDash p$. In this diagram it is also the case that $\pi, 0 \vDash (p \vee q)$.

Although a bit unintuitive at first, it will be convenient to convert all finite traces into infinite ones. To do the conversion, we simply envision that once the state-transition system (aka automaton) reaches its ostensibly last (nth) state, it languishes furiously in that state over and over:

Definition 3. Stutter-Extend*: The finite trace* $\pi_0, \ldots, \pi_n$ *can be stutter-extended to the infinite trace* $\pi_0, \ldots, \pi_n, \pi_n, \pi_n, \ldots$.

Note that this convention precludes assuming that something must change between one state and the next, which is plausible when we are modeling asynchronous systems.

Here are more examples of computations interpreted in graphical form:

`figs/always_p_timeline.png`

Figure 3: A generic computation π satisfying $\pi, i \vDash \Box p$. Note that p is true at time i at all later times.

figs/eventually_q_timeline.png

Figure 4: A generic computation π satisfying $\pi, i \vDash \Diamond q$. Note that q is true in some state k after state i.

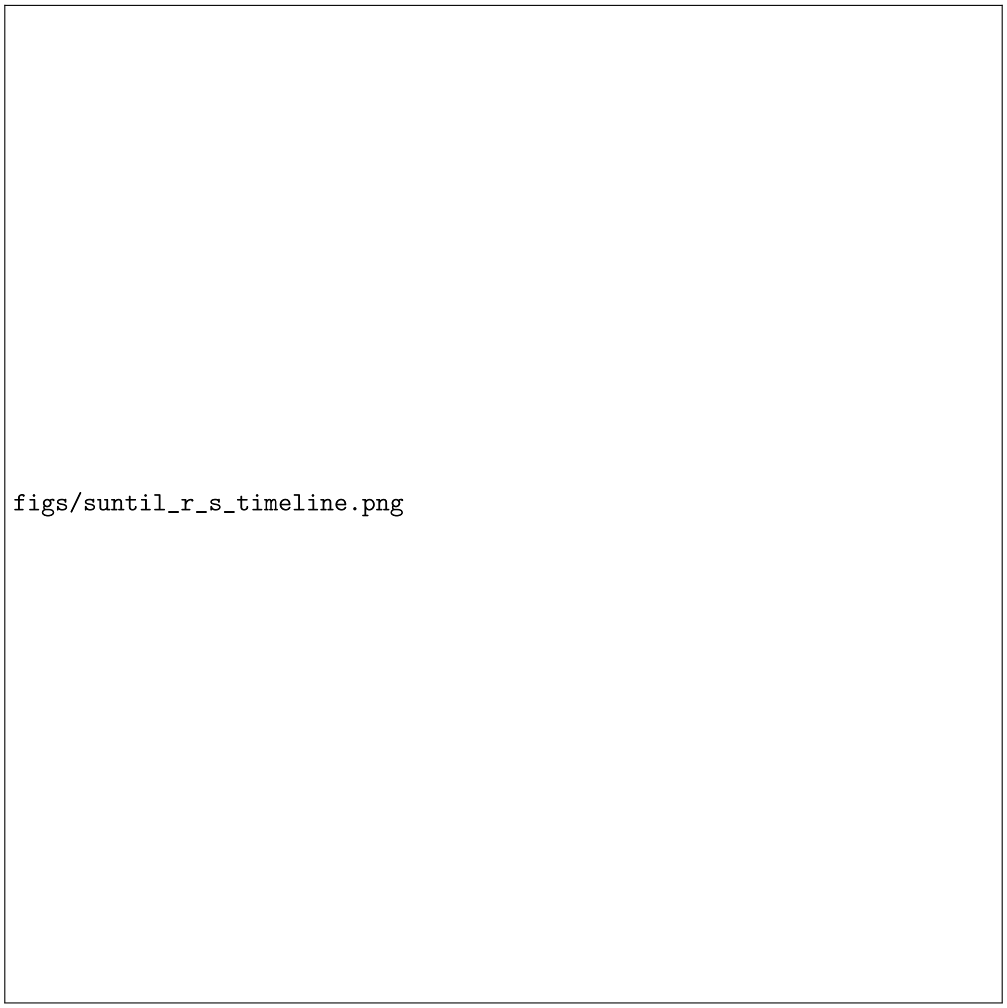


Figure 5: A generic computation π satisfying $\pi, i \vDash r \ \mathcal{U} \ s$. Note that r is true in states i through $(k-1)$.

3.1. Exercise [45 points (3 points each)]

Decide whether each of these formulas is true or false for the following computation π:

figs/always_if_p_q_timeline.png

Figure 6: A timing diagram for a computation π, and propositions p and q.

1. $\pi, 9 \models \Box q$
2. $\pi, 9 \models \Box \neg p$
3. $\pi, 0 \models \Box q$
4. $\pi, 0 \models \Box(p \rightarrow q)$

5. $\pi, 9 \models \Diamond p$
6. $\pi, 9 \models \Diamond \neg p$
7. $\pi, 0 \models \Diamond q$
8. $\pi, 0 \models \Diamond \neg q$
9. $\pi, 0 \models \Diamond (q \rightarrow p)$
10. $\pi, 9 \models \Diamond (q \rightarrow p)$
11. $\pi, 0 \models \Diamond \neg (p \rightarrow q)$
12. $\pi, 0 \models (q \, \mathcal{U} \, p)$
13. $\pi, 2 \models (q \, \mathcal{U} \, p)$
14. $\pi, 5 \models (q \, \mathcal{U} \, p)$
15. $\pi, 9 \models (q \, \mathcal{U} \, p)$

3.2. Exercise [6 points (3 points each)]

Give an English translation of the following LTL formulas. Try to give a natural wording for each, not just a transliteration of the logical operators.

1. $(\Diamond r \rightarrow (p \, \mathcal{U} \, r))$
2. $\Box (q \rightarrow \Box \neg p)$

3.3. Exercise [78 points (6 points each)]

In the following, give an LTL formula that formalizes the given English wording. If the English is subject to any ambiguity, as it frequently is, describe how you are disambiguating it, and why. *Hint:* it helps to draw timelines first and then check your formulas against the timelines to make sure they describe exactly the set of timelines you envision.

1. p is true.
2. p becomes true before q.
3. p will happen at most once.
4. p will happen at most twice.
5. The light always blinks. Use the following proposition: p = the light is on. (You may not make any assumptions as to the number of time steps the light is on or off during any single 'blink,' only 'blinking' constitutes a continuous series of changes of state between 'on' and 'off.')
6. The lights of a traffic signal always light in the following sequence: green, yellow, red, and back to green, etc., with exactly one light on at any time. Use the following propositions: g = the green light is on, y = the yellow light is on, and r = the red light is on.
7. Whenever a train is passing, the gate is down. (Use the following propositions: p = train is passing; g = gate is down.)
8. If a train is approaching or passing, then the light is flashing. (Use the following propositions: a = train is approaching; p = train is passing; l = light is flashing.)
9. If the gate is up and the light is not flashing, then no train is passing or approaching. (Use the following propositions: g = gate is down; l = light is flashing; a = train is approaching; p = train is passing.)

10. If a train is approaching, the gate will be down before the next train passes.

figs/train_timeline1.png

Figure 7: A timing diagram for a possible computation π with propositions a, p, and g illustrating traces that satisfy the requirement "if a train is approaching, the gate will be down before the next train passes."

11. If a train has finished passing, then later the gate will be up.

figs/train_timeline2.png

Figure 8: A timing diagram for a possible computation π with propositions p, and g illustrating traces that satisfy the requirement “if a train has finished passing, then later the gate will be up.”

12. The gate will be up infinitely many times.
13. If a train is approaching, then it will be passing, and later it will be done passing with no train approaching.

figs/train_timeline3.png

Figure 9: A timing diagram for a possible computation π with propositions a, and p illustrating traces that satisfy the requirement "if a train is approaching, then it will be passing, and later it will be done passing with no train approaching."

The Dining Philosophers Problem is a classic conundrum in mathematics. The idea is that a group of N philosophers sit at a single round table for dinner. There are N forks, one placed between each plate. To successfully eat each bite, a philosopher needs both of the adjacent forks. Thus, as one consequence, two adjacent philosophers cannot eat at the same time, since they cannot both have the fork in between them at the same time. The question is what strategies can the philosophers have such that each philosopher eventually eats. Typically, but not necessarily, we also require that each philosopher has the same strategy. Yes, this is a very silly way to eat, which is why some instances of the problem use chop sticks instead of forks as one clearly needs two chopsticks to effectively leverage them as utensils.

Linear Temporal Logic (LTL) formulas reason about linear timelines:

- a finite set *Prop* of atomic propositions
- Boolean connectives: $\neg$, $\wedge$, $\vee$, and $\rightarrow$
- temporal connectives:

$\mathcal{X}\varphi$	next time
$\varphi\ \mathcal{U}\ \psi$	until
$\varphi\ \mathcal{R}\ \psi$	release
$\Box\varphi$	also called $\mathcal{G}$ for "globally"
$\Diamond\varphi$	also called $\mathcal{F}$ for "in the future"

Computational Tree Logic (CTL) reasons about branching paths:

- temporal connectives are always proceeded by path quantifiers:

$\mathcal{A}$	for all paths
$\mathcal{E}$	exists a path

Figure 10: Syntax of LTL and CTL

3.4. Exercise [30 points (6 points each)]

Using temporal logic, formally specify the following desired properties of solutions to the Dining Philosophers Problem. Use the following logic variables, where $0 \leq i < N$:

- l_i: Philosopher i has his/her left fork.
- r_i: Philosopher i has his/her right fork.

For each question, your answer should cover exactly the given condition – nothing more or less. You may assume $N = 3$.

1. No fork is ever claimed to be held by two philosophers simultaneously.
2. Philosopher i gets to eat (at least once).
3. Each philosopher gets to eat infinitely often.
4. The philosophers don't deadlock. (The main difficulty is to conceptualize and restate "deadlock" within this specific model in terms of the available logic variables.) You may assume philosophers pick up two forks in some order, eat, and drop both forks. For example, one might pick up a single fork and then drop it. Or, the philosophers might be lazy and never pick up a fork.
5. Describe a Dining Philosophers Problem run in which philosophers don't deadlock, but it is not the case that each philosopher gets to eat infinitely often.

4. Computational Tree Logic

Computational Tree Logic (CTL) is a branching time logic that reasons over many possible traces through time. Historically, CTL was the first logic used in model checking [**?**] and it remains a popular specification logic. Unlike LTL, for which every time instance has exactly one immediate successor, in CTL a time instance has a finite, non-zero number of immediate successors. A branching timeline starts in the current time step, and may progress to any one of potentially many possible infinite futures. In addition to reasoning along a timeline, as we did for linear time logic, branching time temporal operators must also reason across the possible branches. Consequently, the temporal operators in CTL are all two-part operators with one part specifying, similarly to LTL, the action to occur along a future timeline and another part specifying whether this action takes place on at least one branch or all branches.

Computational Tree Logic (CTL) formulas are composed of a finite set *Prop* of atomic propositions, the Boolean connectives $\neg$, $\wedge$, $\vee$, and $\rightarrow$, and indivisible quantifier pairings of the path quantifiers $\mathcal{A}$ (always, for all paths), and $\mathcal{E}$ (there exists a path), with the linear temporal connectives of LTL: $\mathcal{U}$ (until), $\mathcal{X}$ (also called $\bigcirc$ for "next time"), $\Box$ (also called $\mathcal{G}$ for "globally") and $\Diamond$ (also called $\mathcal{F}$ for "in the future"). We define CTL formulas inductively:

Definition 4. *For every $p \in Prop$, p is a formula. If φ and ψ are formulas, then so are:*

$\neg\varphi$	$\varphi \wedge \psi$	$\varphi \vee \psi$	$\varphi \rightarrow \psi$	$\mathcal{A}(\varphi\ \mathcal{U}\ \psi)$	$\mathcal{E}(\varphi\ \mathcal{U}\ \psi)$
$\mathcal{A}\mathcal{X}\varphi$	$\mathcal{E}\mathcal{X}\varphi$	$\mathcal{A}\Box\varphi$	$\mathcal{E}\Box\varphi$	$\mathcal{A}\Diamond\varphi$	$\mathcal{E}\Diamond\varphi$

CTL formulas describe the behavior of the variables in *Prop* over a branching series of time steps starting at time zero and extending infinitely into the future. As for LTL, *computations* are functions that assign truth values to the elements of *Prop* at each time instant.

Definition 5. *We interpret CTL formulas over sets of possible computations of the form $\pi : \omega \rightarrow 2^{Prop}$. Due to the branching nature of the logic, there may be many future paths possible from any one time step: $\pi_0, \pi_1, \pi_2, \ldots$ We will call the set of all of the possible paths from the current time step Π. We define $\Pi, i \vDash \varphi$ (set of possible computations $\pi_{0\ldots m} \in \Pi$ at time instant $i \in \omega$ satisfies CTL formula φ) as follows:*

- $\Pi, i \vDash p$ *for* $p \in Prop$ *if* $p \in \Pi(i)$.
- $\Pi, i \vDash \neg\varphi$ *if* $\Pi, i \nvDash \varphi$.
- $\Pi, i \vDash \varphi \wedge \psi$ *if* $\Pi, i \vDash \varphi$ *and* $\Pi, i \vDash \psi$.
- $\Pi, i \vDash \varphi \vee \psi$ *if* $\Pi, i \vDash \varphi$ *or* $\Pi, i \vDash \psi$.
- $\Pi, i \vDash \mathcal{A}\mathcal{X}\varphi$ *if* $(\forall n)\pi_n, i+1 \vDash \varphi$.
- $\Pi, i \vDash \mathcal{E}\mathcal{X}\varphi$ *if* $(\exists n)\pi_n, i+1 \vDash \varphi$.
- $\Pi, i \vDash \mathcal{A}(\varphi\ \mathcal{U}\ \psi)$ *if* $(\forall n)(\exists j \geq i)$, *such that* $\pi_n, j \vDash \psi$ *and* $\forall k, i \leq k < j$, *we have* $\pi_n, k \vDash \varphi$.
- $\Pi, i \vDash \mathcal{E}(\varphi\ \mathcal{U}\ \psi)$ *if* $(\exists n)(\exists j \geq i)$, *such that* $\pi_n, j \vDash \psi$ *and* $\forall k, i \leq k < j$, *we have* $\pi_n, k \vDash \varphi$.
- $\Pi, i \vDash \mathcal{A}\Box\varphi$ *if* $(\forall n)(\forall j \geq i)$, $\pi_n, j \vDash \varphi$.
- $\Pi, i \vDash \mathcal{E}\Box\varphi$ *if* $(\exists n)(\forall j \geq i)$, $\pi_n, j \vDash \varphi$.
- $\Pi, i \vDash \mathcal{A}\Diamond\varphi$ *if* $(\forall n)(\exists j \geq i)$, *such that* $\pi_n, j \vDash \varphi$.
- $\Pi, i \vDash \mathcal{E}\Diamond\varphi$ *if* $(\exists n)(\exists j \geq i)$, *such that* $\pi_n, j \vDash \varphi$.

Equivalences. The path quantifiers $\mathcal{A}$ and $\mathcal{E}$, representing universal and existential quantification over the branching paths are duals and can be defined in terms of each other. Similarly, $\Box$ and $\Diamond$ signify universal and existential quantification over time steps along a single paths and are therefore also duals. (Again, the $\mathcal{U}$-operator is *strong until* because its second argument *must* be true at some time.) These realizations lead us to the following set of semantic equivalences:

- $(\mathcal{A}\Diamond\varphi) \equiv (\neg\mathcal{E}\Box(\neg\varphi)) \equiv (\mathcal{A}(\mathit{true}\ \mathcal{U}\ \varphi))$.
- $(\mathcal{A}\Box\varphi) \equiv (\neg\mathcal{E}\Diamond(\neg\varphi)) \equiv (\neg\mathcal{E}(\mathit{true}\ \mathcal{U}\ \neg\varphi))$.
- $(\mathcal{E}\Diamond\varphi) \equiv (\neg\mathcal{A}\Box(\neg\varphi)) \equiv (\mathcal{E}(\mathit{true}\ \mathcal{U}\ \varphi)$.
- $(\mathcal{E}\Box\varphi) \equiv (\neg\mathcal{A}\Diamond(\neg\varphi)) \equiv (\neg\mathcal{A}(\mathit{true}\ \mathcal{U}\ \neg\varphi))$.
- $(\mathcal{A}\mathcal{X}\varphi) \equiv (\neg\mathcal{E}\mathcal{X}(\neg\varphi))$.
- $(\mathcal{E}\mathcal{X}\varphi) \equiv (\neg\mathcal{A}\mathcal{X}(\neg\varphi))$.
- $(\mathcal{A}[\varphi\ \mathcal{U}\ \psi]) \equiv (\neg\mathcal{E}[(\neg\psi)\ \mathcal{U}\ (\neg\varphi \wedge \neg\psi)]) \wedge (\neg\mathcal{E}\Box(\neg\psi))$.

4.1. Exercise [60 points (6 points each)]

In the following, give a CTL formula that formalizes the given English wording. If the English is subject to any ambiguity, as it frequently is, describe how you are disambiguating it, and why. *Hint:* it helps to draw pictures of time as a branching tree first and then check your formulas against your diagrams to make sure they describe exactly the set of branching traces you envision.

1. If we remove the support, necessarily next the structure will fall. (Hint: define a Boolean variable like *remove* to indicate whether the support has been removed and another Boolean variable like $fall$ to indicate a fall occurring.)
2. The system cannot be one way and immediately next time be the other way. (Variables: one_way, $other\_way$)
3. If the plane takes off it will certainly land. (Variables: $takeoff$, $land$)
4. If the GPS fails, the UAS may possibly eventually become lost. (Variables: $fail$, $lost$)
5. The aircraft will inevitably land. (Variable: $land$)
6. It might be the case that the air traffic controller sends a resolution next time. (Variable: $resolution$)
7. Our Mars rover may possibly henceforth be buried in the sand. (Variable: $buried$)
8. We may possibly have to wait until the next wind storm clears the sand to communicate with our Mars rover.
9. To resolve a TSAFE alert between Aircraft_1 and Aircraft_2, there could be a resolution maneuver generated for either Aircraft_1, or Aircraft_2, or both. (Variables: $alert$, $res1$, $res2$)
10. If there is an AutoResolver alert between Aircraft_1 and Aircraft_2, and there are no more imminent alerts, there is always an option where the controller permits the AutoResolver resolution.

5. CTL*

Emerson and Halpern first invented the logic CTL* in 1983 [**?**]. This logic combines the syntaxes of the two logics LTL and CTL. It includes all of the logical operators in LTL and both path quantifiers of CTL but does not have the CTL restriction that temporal operators must appear in pairs. Both LTL and CTL are proper subsets of CTL*, as are all combinations of LTL and CTL formulas; CTL* is more expressive than both LTL and CTL combined. For example, the formulas $E(\Box\Diamond p)$ and $A(\Diamond\Box p) \vee A\Box(E\Diamond p)$ are both in CTL* but in neither LTL nor CTL.

However, this expressive power comes at a great cost. The model-checking problem for CTL* is PSPACE-complete [**?**], which the same general complexity as for LTL, though the algorithm is considerably more complex to implement and there are currently no model checkers for this logic. Indeed, for practical model-checking problems such as compositional verification and verification of open or reactive systems (i.e. those systems that interact with an environment) CTL* is dominated by LTL [**?**]. Furthermore, the simple task of specification debugging via satisfiability checking is 2EXPTIME-complete for CTL* [**? ?**]. Simply translating a CTL* specification into an automaton for model checking involves a doubly-exponential blow-up [**?**]. So, despite the deceptive time complexity for the general model-checking problem, adding branching to LTL is not free. In practice, should LTL prove to be too limited to express a desired property, CTL* is almost certainly sufficient [**?**]. However, the lack of industrial model-checking tools that accept CTL* specifications is a deterrent to the use of this logic.