Lecture 3
Linear Temporal Logic (LTL)

Richard M. Murray
Nok Wongpiromsarn    Ufuk Topcu
California Institute of Technology

AFRL, 24 April 2012

Outline
• Syntax and semantics of LTL
• Specifying properties in LTL
• Equivalence of LTL formulas
• Fairness in LTL
• Other temporal logics (if time)

Chapter 5
Formal Methods for System Verification

Specification using LTL
- Linear temporal logic (LTL) is a mathematical language for describing linear-time properties
- Provides a particularly useful set of operators for constructing LT properties without specifying sets

Methods for verifying an LTL specification
- *Theorem proving*: use formal logical manipulations to show that a property is satisfied for a given system model
- *Model checking*: explicitly check all possible executions of a system model and verify that each of them satisfies the formal specification
  - Roughly like trying to prove stability by simulating every initial condition
  - Works because discrete transition systems have finite number of states
  - Very good tools now exist for doing this efficiently (SPIN, nuSMV, etc)
Temporal Logic Operators

Two key operators in temporal logic
- ◊ “eventually” – a property is satisfied at some point in the future
- □ “always” – a property is satisfied now and forever into the future

“Temporal” refers underlying nature of time
- Linear temporal logic ⇒ each moment in time has a well-defined successor moment
- Branching temporal logic ⇒ reason about multiple possible time courses
- “Temporal” here refers to “ordered events”; no explicit notion of time

LTL = linear temporal logic
- Specific class of operators for specifying linear time properties
- Introduced by Pneuli in the 1970s (recently passed away)
- Large collection of tools for specification, design, analysis

Other temporal logics
- CTL = computation tree logic (branching time; will see later, if time)
- TCTL = timed CTL - check to make sure certain events occur in a certain time
- TLA = temporal logic of actions (Lamport) [variant of LTL]
- μ calculus = for reactive systems; add “least fixed point” operator (more tomorrow)
Syntax of LTL

LTL formulas:

\[ \varphi ::= \text{true} \mid a \mid \varphi_1 \land \varphi_2 \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi_1 \mathbf{U} \varphi_2 \]

- \( a \) = atomic proposition
- \( \bigcirc \) = “next”: \( \varphi \) is true at next step
- \( \mathbf{U} \) = “until”: \( \varphi_2 \) is true at some point, \( \varphi_1 \) is true until that time

Operator precedence

- Unary bind stronger than binary
- \( \mathbf{U} \) takes precedence over \( \land, \lor \) and \( \rightarrow \)

Formula evaluation: evaluate LTL propositions over a sequence of states (path):

- Same notation as linear time properties: \( \sigma \models \varphi \) (path “satisfies” specification)
Additional Operators and Formulas

“Primary” temporal logic operators

- Eventually $\diamond \phi := true U \phi$  \hspace{1cm} $\phi$ will become true at some point in the future
- Always $\square \phi := \neg \diamond \neg \phi$  \hspace{1cm} $\phi$ is always true; “(never (eventually (\neg \phi)))”

Some common composite operators

- $p \to \diamond q$  \hspace{1cm} $p$ implies eventually $q$ (response)
- $p \to q U r$  \hspace{1cm} $p$ implies $q$ until $r$ (precedence)
- $\square \diamond p$  \hspace{1cm} always eventually $p$ (progress)
- $\diamond \square p$  \hspace{1cm} eventually always $p$ (stability)
- $\diamond p \to \diamond q$  \hspace{1cm} eventually $p$ implies eventually $q$ (correlation)

Operator precedence

- Unary binds stronger than binary
- Bind from right to left:
  $\square \diamond p = (\square (\diamond p))$
  $p U q U r = p U (q U r)$
- $U$ takes precedence over $\land, \lor$ and $\to$
**Example: Traffic Light**

**System description**
- Focus on lights in a particular direction
- Light can be any of three colors: green, yellow, red
- Atomic propositions = light color

**Ordering specifications**
- Liveness: "traffic light is green infinitely often"
  \[ \square \Diamond \text{green} \]
- Chronological ordering: "once red, the light cannot become green immediately"
  \[ \square (\text{red} \rightarrow \neg \Diamond \text{green}) \]
- More detailed: "once red, the light always becomes green eventually after being yellow for some time"
  \[ \square (\text{red} \rightarrow (\Diamond \text{green} \land (\neg \text{green} \lor \Diamond \text{yellow}))) \]
  \[ \square (\text{red} \rightarrow \Diamond (\text{red} \lor (\text{yellow} \land \Diamond (\text{yellow} \lor \Diamond \text{green})))) \]

**Progress property**
- Every request will eventually lead to a response
  \[ \square (\text{request} \rightarrow \Diamond \text{response}) \]
Semantics: when does a path satisfy an LTL spec?

Definition 5.6. Semantics of LTL (Interpretation over Words)
Let \( \varphi \) be an LTL formula over \( \mathcal{AP} \). The LT property induced by \( \varphi \) is

\[
\text{Words}(\varphi) = \left\{ \sigma \in (2^{\mathcal{AP}})^\omega \mid \sigma \models \varphi \right\}
\]

where the satisfaction relation \( \models \subseteq (2^{\mathcal{AP}})^\omega \times \text{LTL} \) is the smallest relation with the properties in Figure 5.2.

\[
\begin{align*}
\sigma & \models \text{true} \\
\sigma & \models a \quad \text{iff} \quad a \in A_0 \quad \text{(i.e.,} \quad A_0 \models a) \\
\sigma & \models \varphi_1 \land \varphi_2 \quad \text{iff} \quad \sigma \models \varphi_1 \quad \text{and} \quad \sigma \models \varphi_2 \\
\sigma & \models \neg \varphi \quad \text{iff} \quad \sigma \not\models \varphi \\
\sigma & \models \Diamond \varphi \quad \text{iff} \quad \exists j \geq 0. \sigma[j \ldots] \models \varphi \\
\sigma & \models \Box \varphi \quad \text{iff} \quad \forall j \geq 0. \sigma[j \ldots] \models \varphi \\
\sigma & \models \varphi_1 \lor \varphi_2 \quad \text{iff} \quad \exists j \geq 0. \sigma[j \ldots] \models \varphi_2 \quad \text{and} \quad \sigma[i \ldots] \models \varphi_1, \text{ for all } 0 \leq i < j
\end{align*}
\]

Figure 5.2: LTL semantics (satisfaction relation \( \models \)) for infinite words over \( 2^{\mathcal{AP}} \).
Semantics of LTL

The semantics of the combinations of $\square$ and $\Diamond$ can now be derived:

$$\sigma \models \square \Diamond \varphi \iff \exists j. \sigma[j \ldots] \models \varphi$$

$$\sigma \models \Diamond \square \varphi \iff \forall j. \sigma[j \ldots] \models \varphi.$$ 

Here, $\exists j$ means $\forall i \geq 0. \exists j \geq i$, “for infinitely many $j \in \mathbb{N}$”, while $\forall j$ stands for $\exists i \geq 0. \forall j \geq i$, “for almost all $j \in \mathbb{N}$”.

Definition 5.7.  Semantics of LTL over Paths and States

Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system without terminal states, and let $\varphi$ be an LTL-formula over $AP$.

- For infinite path fragment $\pi$ of $TS$, the satisfaction relation is defined by
  $$\pi \models \varphi \iff \text{trace}(\pi) \models \varphi.$$ 

- For state $s \in S$, the satisfaction relation $\models$ is defined by
  $$s \models \varphi \iff (\forall \pi \in \text{Paths}(s). \pi \models \varphi).$$

- $TS$ satisfies $\varphi$, denoted $TS \models \varphi$, if $\text{Traces}(TS) \subseteq \text{Words}(\varphi)$. 
Semantics of LTL

From this definition, it immediately follows that

\[ TS \models \varphi \]

iff

\[ \text{Traces}(TS) \subseteq \text{Words}(\varphi) \]

iff

\[ TS \models \text{Words}(\varphi) \]

iff

\[ \pi \models \varphi \text{ for all } \pi \in \text{Paths}(TS) \]

iff

\[ s_0 \models \varphi \text{ for all } s_0 \in I. \]

(* Definition 5.7 *)

(* Definition of \( \models \) for LT properties *)

(* Definition of \( \text{Words}(\varphi) \) *)

(* Definition 5.7 of \( \models \) for states *)

Remarks

- Which condition you use depends on type of problem under consideration
- For reasoning about correctness, look for (lack of) intersection between sets:
Consider the following transition system

Consider the transition system $TS$ depicted in Figure 5.3 with the set of propositions $AP = \{a, b\}$. For example, we have that $TS \models \Box a$, since all states are labeled with $a$, and hence, all traces of $TS$ are words of the form $A_0 A_1 A_2 \ldots$ with $a \in A_i$ for all $i \geq 0$. Thus, $s_i \models \Box a$ for $i = 1, 2, 3$. Moreover:

$s_1 \models \O (a \land b)$ since $s_2 \models a \land b$ and $s_2$ is the only successor of $s_1$

$s_2 \not\models \O (a \land b)$ and $s_3 \not\models \O (a \land b)$ as $s_3 \in Post(s_2), s_3 \in Post(s_3)$ and $s_3 \not\models a \land b$.

This yields $TS \not\models \O (a \land b)$ as $s_3$ is an initial state for which $s_3 \not\models \O (a \land b)$. As another example:

$TS \models \Box (\neg b \rightarrow \Box (a \land \neg b))$, 

since $s_3$ is the only $\neg b$ state, $s_3$ cannot be left anymore, and $a \land \neg b$ in $s_3$ is true. However,

$TS \not\models b \mathcal{U} (a \land \neg b)$,

since the initial path $(s_1 s_2)^\omega$ does not visit a state for which $a \land \neg b$ holds. Note that the initial path $(s_1 s_2)^* s_3^\omega$ satisfies $b \mathcal{U} (a \land \neg b)$. 

$\blacksquare$
Specifying Timed Properties for Synchronous Systems

For \textit{synchronous} systems, LTL can be used as a formalism to specify “real-time” properties that refer to a discrete time scale. Recall that in synchronous systems, the involved processes proceed in a lock step fashion, i.e., at each discrete time instance each process performs a (sometimes idle) step. In this kind of system, the next-step operator $\bigcirc$ has a “timed” interpretation: $\bigcirc \varphi$ states that “at the next time instant $\varphi$ holds”. By putting applications of $\bigcirc$ in sequence, we obtain, e.g.:

$$\bigcirc_k \varphi \overset{\text{def}}{=} \underbrace{\bigcirc \bigcirc \ldots \bigcirc}_{k\text{-times}} \varphi$$

“$\varphi$ holds after (exactly) $k$ time instants”.

Assertions like “$\varphi$ will hold within at most $k$ time instants” are obtained by

$$\lozenge \leq_k \varphi = \bigvee_{0 \leq i \leq k} \bigcirc^i \varphi.$$ 

Statements like “$\varphi$ holds now and will hold during the next $k$ instants” can be represented as follows:

$$\square \leq_k \varphi = \neg \lozenge \leq_k \neg \varphi = \neg \bigvee_{0 \leq i \leq k} \bigcirc^i \neg \varphi.$$ 

Remark

- Idea can be extended to non-synchronous case (eg, Timed CTL [later])
Equivalence of LTL Formulas

Definition 5.17. Equivalence of LTL Formulae

LTL formulae \( \varphi_1, \varphi_2 \) are equivalent, denoted \( \varphi_1 \equiv \varphi_2 \), if \( \text{Words}(\varphi_1) = \text{Words}(\varphi_2) \).

<table>
<thead>
<tr>
<th>Duality Law</th>
<th>Idempotency Law</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg \lozenge \varphi \equiv \lozenge \neg \varphi )</td>
<td>( \lozenge \lozenge \varphi \equiv \lozenge \varphi )</td>
</tr>
<tr>
<td>( \neg \lozenge \varphi \equiv \Box \neg \varphi )</td>
<td>( \Box \Box \varphi \equiv \Box \varphi )</td>
</tr>
<tr>
<td>( \neg \Box \varphi \equiv \lozenge \neg \varphi )</td>
<td>( \varphi \lor (\varphi \lor \psi) \equiv \varphi \lor \psi )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Absorption Law</th>
<th>Expansion Law</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lozenge \Box \lozenge \varphi \equiv \Box \lozenge \varphi )</td>
<td>( \varphi \lor \psi \equiv \psi \lor (\varphi \lor \Box (\varphi \lor \psi)) )</td>
</tr>
<tr>
<td>( \Box \lozenge \varphi \equiv \lozenge \Box \varphi )</td>
<td>( \lozenge \psi \equiv \psi \lor \Box \lozenge \psi )</td>
</tr>
<tr>
<td>( \Box \Box \varphi \equiv \Box \varphi )</td>
<td>( \Box \psi \equiv \psi \land \Box \Box \psi )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distributive Law</th>
<th>Non-identities</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lozenge (\varphi \lor \psi) \equiv (\lozenge \varphi) \lor (\lozenge \psi) )</td>
<td>( \lozenge (a \land b) \neq \lozenge a \land \lozenge b )</td>
</tr>
<tr>
<td>( \lozenge (\varphi \land \psi) \equiv \lozenge \varphi \land \lozenge \psi )</td>
<td>( \Box (a \lor b) \neq \Box a \lor \Box b )</td>
</tr>
<tr>
<td>( \Box (\varphi \land \psi) \equiv \Box \varphi \land \Box \psi )</td>
<td></td>
</tr>
</tbody>
</table>
LTL Specs for Control Protocols: RoboFlag Drill

Task description
- Incoming robots should be blocked by defending robots
- Incoming robots are assigned randomly to whoever is free
- Defending robots must move to block, but cannot run into or cross over others
- Allow robots to communicate with left and right neighbors and switch assignments

Goals
- Would like a provably correct, distributed protocol for solving this problem
- Should (eventually) allow for lost data, incomplete information

Questions
- How do we describe task in terms of LTL?
- Given a protocol, how do we prove specs?
- How do we design the protocol given specs?
Properties for RoboFlag program

CCL formulas (will cover in more detail later)

- $q'$ → $\diamond q$: evaluate $q$ at the next action in path
- $p \rightarrow q$ → $\square (p \rightarrow \diamond q)$: "$p$ leads to $q$": if $p$ is true, $q$ will eventually be true
- $p \land q$ → $\land (p \rightarrow \diamond q)$: if $p$ is true, then next time state changes, $q$ will be true

Safety (Defenders do not collide)

$$z_i < z_{i+1} \land z_i < z_{i+1}$$

True if robots $i$ and $i+1$ have targets that cause crossed paths

Stability (switch predicate stays false)

$$\forall i . y_i > 2\delta \land z_i + 2\delta < z_{i+1} \land \neg switch_{i,i+1} \land \land switch_{i,i+1}$$

Robots are "far enough" apart.

“Lyapunov” stability

- Remains to show that we actually approach the goal (robots line up with targets)
- Will see later we can do this using a Lyapunov function
Fairness

Mainly an issue with concurrent processes

- To make sure that the proper interaction occurs, often need to know that each process gets executed reasonably often
- Multi-threaded version: each thread should receive some fraction of processes time

Two issues: implementation and specification

- Q1: How do we implement our algorithms to insure that we get “fairness” in execution
- Q2: how do we model fairness in a formal way to reason about program correctness

Example: Fairness in RoboFlag Drill

- To show that algorithm behaves properly, need to know that each agent communicates with neighbors regularly (infinitely often), in each direction

Difficulty in describing fairness depends on the logical formalism

- Turns out to be pretty easy to describe fairness in linear temporal logic
- Much more difficult to describe fairness for other temporal logics (eg, CTL & variants)
Fairness Properties in LTL

Definition 5.25  LTL Fairness Constraints and Assumptions

Let $\Phi$ and $\Psi$ be propositional logical formulas over a set of atomic propositions

1. An **unconditional LTL fairness constraint** is an LTL formula of the form $ufair = \Box \Diamond \Psi$.

2. A **strong LTL fairness condition** is an LTL formula of the form $sfair = \Box \Diamond \Phi \rightarrow \Box \Diamond \Psi$.

3. A **weak LTL fairness constraint** is an LTL formula of the form $wfair = \Diamond \Box \Phi \rightarrow \Box \Diamond \Psi$.

An **LTL fairness assumption** is a conjunction of LTL fairness constraints (of any arbitrary type).

$$fair = ufair \land sfair \land wfair.$$ 

**Rules of thumb**

- strong (or unconditional) fairness: useful for solving contentions
- weak fairness: sufficient for resolving the non-determinism due to interleaving.
**Fairness Properties in LTL**

**Fair paths and traces**

\[
\text{FairPaths}(s) = \{ \pi \in \text{Paths}(s) \mid \pi \models \text{fair} \}, \\
\text{FairTraces}(s) = \{ \text{trace}(\pi) \mid \pi \in \text{FairPaths}(s) \}.
\]

**Definition 5.26. Satisfaction Relation for LTL with Fairness**

For state \( s \) in transition system \( TS \) (over \( AP \)) without terminal states, LTL formula \( \varphi \), and LTL fairness assumption \( \text{fair} \) let

\[
s \models_{\text{fair}} \varphi \iff \forall \pi \in \text{FairPaths}(s). \pi \models \varphi \quad \text{and} \\
TS \models_{\text{fair}} \varphi \iff \forall s_0 \in I. s_0 \models_{\text{fair}} \varphi.
\]

\[\square\]

**Theorem 5.30. Reduction of \( \models_{\text{fair}} \) to \( \models \)**

For transition system \( TS \) without terminal states, LTL formula \( \varphi \), and LTL fairness assumption \( \text{fair} \):

\[
TS \models_{\text{fair}} \varphi \quad \text{if and only if} \quad TS \models (\text{fair} \rightarrow \varphi).
\]
Branching Time and Computational Tree Logic

Consider transition systems with multiple branches

- Eg, nondeterministic finite automata (NFA), nondeterministic Bucchi automata (NBA)
- In this case, there might be multiple paths from a given state
- Q: in evaluating a temporal logic property, which execution branch to we check?

Computational tree logic: allow evaluation over some or all paths

\[ s \models \exists \varphi \iff \pi \models \varphi \text{ for some } \pi \in \text{Paths}(s) \]
\[ s \models \forall \varphi \iff \pi \models \varphi \text{ for all } \pi \in \text{Paths}(s) \]
Example: Triply Redundant Control Systems

Systems consists of three processors and a single voter

- \( s_{i,j} = i \) processors up, \( j \) voters up
- Assume processors fail one at a time; voter can fail at any time
- If voter fails, reset to fully functioning state (all three processors up)
- System is operation if at least 2 processors remain operational

Properties we might like to prove

<table>
<thead>
<tr>
<th>Property</th>
<th>Formalization in CTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possibly the system never goes down</td>
<td>( \exists \square \neg \text{down} )</td>
</tr>
<tr>
<td>Invariantly the system never goes down</td>
<td>( \forall \square \neg \text{down} )</td>
</tr>
<tr>
<td>It is always possible to start as new</td>
<td>( \forall \square \exists \diamond \text{up}_3 )</td>
</tr>
<tr>
<td>The system always eventually goes down and is operational until going down</td>
<td>( \forall ((\text{up}_3 \lor \text{up}_2) \cup \text{down}) )</td>
</tr>
</tbody>
</table>
Other Types of Temporal Logic

**CTL ≠ LTL**
- Can show that LTL and CTL are not proper subsets of each other
- LTL reasons over a complete path; CTL from a given state

**CTL** captures both

\[ \Phi ::= \text{true} \mid a \mid \Phi_1 \land \Phi_2 \mid \neg \Phi \mid \exists \varphi \]

**Timed Computational Tree Logic**
- Extend notions of transition systems and CTL to include “clocks” (multiple clocks OK)
- Transitions can depend on the value of clocks
- Can require that certain properties happen within a given time window

\[ \forall \square (\text{far} \rightarrow \forall \Diamond \leq 1 \forall \Box \leq 1 \text{ up}) \]
Summary: Specifying Behavior with LTL

Description

- State of the system is a snapshot of values of all variables
- Reason about paths $\sigma$: sequence of states of the system
- No strict notion of time, just ordering of events
- Actions are relations between states: state $s$ is related to state $t$ by action $a$ if $a$ takes $s$ to $t$ (via prime notation: $x' = x + 1$)
- Formulas (specifications) describe the set of allowable behaviors
- Safety specification: what actions are allowed
- Fairness specification: when can a component take an action (e.g., infinitely often)

Example

- Action: $a \equiv x' = x + 1$
- Behavior: $\sigma \equiv x := 1, x := 2, x := 3, \ldots$
- Safety: $\square x > 0$ (true for this behavior)
- Fairness: $\square (x' = x + 1 \lor x' = x) \land \square \diamond (x' \neq x)$

Properties

- Can reason about time by adding “time variables” ($t' = t + 1$)
- Specifications and proofs can be difficult to interpret by hand, but computer tools existing (e.g., TLC, Isabelle, PVS, SPIN, etc)

- $\square p \equiv$ always $p$ (invariance)
- $\diamond p \equiv$ eventually $p$ (guarantee)
- $p \rightarrow \diamond q \equiv$ implies eventually $q$ (response)
- $p \rightarrow q \text{ until } r \equiv$ implies $q$ until $r$ (precedence)
- $\square \diamond p \equiv$ always eventually $p$ (progress)
- $\diamond \square p \equiv$ eventually always $p$ (stability)
- $\diamond p \rightarrow \diamond q \equiv$ eventually $p$ implies eventually $q$ (correlation)