Lecture 3
Automata-Based Representation of Linear-Time Properties and Linear Temporal Logic (LTL)

Richard M. Murray
Nok Wongpiromsarn         Ufuk Topcu

EECI, 14 May 2012

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

Chapter 5
Representations of linear-time properties

Two more representations of linear-time properties:

- Automata-based: readable by machine
- Linear temporal logic (LTL): readable by humans

- LTL is a formal language for describing linear-time properties
- It provides particularly useful operators for constructing linear-time properties without explicitly specifying sets (of, e.g., infinite sequences of subsets of atomic propositions)
Nondeterministic finite automaton (NFA)

A nondeterministic finite automaton \( A = (Q, \Sigma, \delta, Q_0, F) \) is a tuple with
- \( Q \) is a set of states,
- \( \Sigma \) is an alphabet,
- \( \delta : Q \times \Sigma \rightarrow 2^Q \) is a transition function,
- \( Q_0 \subseteq Q \) is a set of initial states, and
- \( F \subseteq Q \) is a set of accept (or: final) states.

Let \( w = A_1 \ldots A_n \in \Sigma^* \) be a finite word. A run for \( w \) in \( A \) is a finite sequence of states \( q_0 q_1 \ldots q_n \) s.t.
- \( q_0 \in Q_0 \)
- \( q_i \xrightarrow{A_{i+1}} q_{i+1} \) for all \( 0 \leq i < n \).

A run \( q_0 q_1 \ldots q_n \) is called accepting if \( q_n \in F \).

A finite word in accepted if it leads to an accepting run.

The accepted language \( \mathcal{L}(A) \) of \( A \) is the set of finite words in \( \Sigma^* \) accepted by \( A \).
Regular safety properties

A set $L \subseteq \Sigma^*$ of finite strings is called a regular language if there is a nondeterministic finite automaton $A$ s.t. $L = L(A)$.

A safety property $P_{safe}$ over $AP$ is called regular if its set of bad prefixes constitutes a regular language over $2^AP$.

That is: $\exists$ NFA $A$ s.t. $L(A) =$ bad prefixes of $P_{safe}$

Example: $AP = \{\text{red}, \text{green}, \text{yellow}\}$
“Each red must be preceded immediately by a yellow” is a regular safety property.

Sample bad prefixes:
- {}{}{red}
- {}{red}
- {yellow}{yellow}{green}{red}
- $A_0 A_1 \ldots A_n$ s.t. $n > 0$, $\text{red} \in A_n$, and $\text{yellow} \notin A_{n-1}$

general form of minimal bad prefixes

NFA: $A = (Q, \Sigma, \delta, Q_0, F)$

language (set of finite words) accepted by the NFA
Verifying regular safety properties

Given a transition system $TS$ and a regular safety property $P_{safe}$, both over the atomic propositions $AP$.

Let $A$ be an NFA s.t. $\mathcal{L}(A) = \text{BadPref}(P_{safe})$.

$TS \models P_{safe}$ iff $\text{Traces}(TS) \subseteq P_{safe}$
iff $\text{Traces}(TS) \cap ((2^AP)^\omega \setminus P_{safe}) = \emptyset$
iff $\text{Traces}(TS) \cap \text{BadPref}(P_{safe}).(2^AP)^\omega = \emptyset$
iff $\text{pref}(\text{Traces}(TS)) \cap \text{BadPref}(P_{safe}) = \emptyset$
iff $\text{pref}(\text{Traces}(TS)) \cap \mathcal{L}(A) = \emptyset$

finite prefixes

For words $w$ and $\sigma$, $w.\sigma$ denotes their concatenation.
<table>
<thead>
<tr>
<th>Invariant</th>
<th>Safety</th>
<th>Liveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>state condition</td>
<td>something bad</td>
<td>something good</td>
</tr>
<tr>
<td></td>
<td>never happens</td>
<td>will happen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eventually</td>
</tr>
<tr>
<td>violated at</td>
<td>any infinite run</td>
<td>violated only by</td>
</tr>
<tr>
<td>individual states</td>
<td>violating the property</td>
<td>infinite runs</td>
</tr>
<tr>
<td></td>
<td>has a finite prefix</td>
<td></td>
</tr>
<tr>
<td>verification: find the</td>
<td>verification: based on</td>
<td>verification:</td>
</tr>
<tr>
<td>reachable states and</td>
<td>nondeterministic finite</td>
<td></td>
</tr>
<tr>
<td>check the invariant</td>
<td>automaton which accepts</td>
<td></td>
</tr>
<tr>
<td>condition</td>
<td>“finite runs”</td>
<td>?</td>
</tr>
</tbody>
</table>
### Nondeterministic Buchi automaton (NBA)

A nondeterministic Buchi automaton is same as an NFA \( A = (Q, \Sigma, \delta, Q_0, F) \) with its runs interpreted differently.

Let \( w = A_1A_2 \ldots \in \Sigma^\omega \) be an infinite string. A *run* for \( w \) in \( A \) is an infinite sequence \( q_0q_1 \ldots \) of states s.t.
- \( q_0 \in Q_0 \) and
- \( q_0 \xrightarrow{A_1} q_1 \xrightarrow{A_2} q_2 \xrightarrow{A_3} \ldots \)

A run is *accepting* if \( q_j \in F \) for infinitely many \( j \).

A string \( w \) is accepted by \( A \) if there is an accepting run of \( w \) in \( A \).

\( \mathcal{L}_\omega(A) \): set of infinite strings accepted by \( A \).

A set of infinite string \( \mathcal{L}_\omega \subseteq \Sigma^\omega \) is called an \( \omega \)-regular language if there is an NBA \( A \) s.t. \( \mathcal{L}_\omega = \mathcal{L}_\omega(A) \).

The NBA on the right accepts the infinite words satisfying the LT property: “infinitely often green.”
**ω-regular properties**

An LT property $P$ over $AP$ is called $ω$-regular if $P$ is an $ω$-regular language over $2^{AP}$.

Invariant, regular safety, and various liveness properties are $ω$-regular.

Let $P$ be an $ω$-regular property and $A$ be an NBA that represents the "bad traces" for $P$.

Basic idea behind model checking $ω$-regular properties:

$$TS \not\models P \quad \text{if and only if} \quad \text{Traces}(TS) \not\subseteq P$$

$$\quad \text{if and only if} \quad \text{Traces}(TS) \cap \left(2^{AP}\setminus P\right) \neq \emptyset$$

$$\quad \text{if and only if} \quad \text{Traces}(TS) \cap \overline{P} \neq \emptyset$$

$$\quad \text{if and only if} \quad \text{Traces}(TS) \cap \mathcal{L}_ω(A) \neq \emptyset$$

NBA: $A = (Q, Σ, δ, Q_0, F)$
<table>
<thead>
<tr>
<th><strong>Invariant</strong></th>
<th><strong>Safety</strong></th>
<th><strong>Liveness</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>state condition</td>
<td>something bad never happens</td>
<td>something good will happen eventually</td>
</tr>
<tr>
<td>violated at individual states</td>
<td>any infinite run violating the property has a finite prefix</td>
<td>violated only by infinite runs</td>
</tr>
<tr>
<td>verification: find the reachable states and check the invariant condition</td>
<td>verification: based on nondeterministic finite automaton which accepts “finite runs”</td>
<td>verification: based on nondeterministic Buchi automaton which accepts infinite runs</td>
</tr>
</tbody>
</table>
Representations of linear-time properties

Two more representations of linear-time properties:

- Automata-based: readable by machine
- Linear temporal logic (LTL): readable by humans

- LTL is a formal language for describing linear-time properties
- It provides particularly useful operators for constructing linear-time properties without explicitly specifying sets (of, e.g., infinite sequences of subsets of atomic propositions)
Temporal logic

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 = add “least fixed point” operator (more tomorrow)
Syntax of LTL

LTL formulas:

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

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

Formula evaluation: evaluate LTL propositions over a sequence of subsets of atomic propositions
Additional operators and formulas

Derived temporal logic operators

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

Some common composite operators

- $p \rightarrow \diamond q$  $p$ implies eventually $q$ (response)
- $p \rightarrow q U r$  $p$ implies $q$ until $r$ (precedence)
- $\Box \diamond p$  always eventually $p$ (progress)
- $\diamond \Box p$  eventually always $p$ (stability)
- $\diamond p \rightarrow \diamond q$  eventually $p$ implies eventually $q$ (correlation)

Operator precedence

- Unary binds stronger than binary
  $\neg \phi_1 U \diamond \phi_2 = (\neg \phi_1)U (\diamond \phi_2)$
- Bind from right to left:
  $\Box \diamond p = (\Box (\diamond p))$
  $p U q U r = p U (q U r)$
- $U$ takes precedence over $\land, \lor$ and $\rightarrow$
**Semantics: when does a path satisfy an LTL spec?**

Let \( \varphi \) be an LTL formula over \( AP \). The linear-time property induced by \( \varphi \) is

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

where the satisfaction relation is the smallest relation with the properties

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

For derived operators:

\[
\begin{align*}
\sigma &\models \Box \varphi = \neg \Diamond \neg \varphi \quad \text{iff} \quad \neg \exists j \geq 0. \sigma[j \ldots] \not\models \varphi \\
\sigma &\models \Box \Diamond \varphi \quad \text{iff} \quad \exists j. \sigma[j \ldots] \models \varphi \\
\sigma &\models \Diamond \Box \varphi \quad \text{iff} \quad \forall j. \sigma[j \ldots] \models \varphi
\end{align*}
\]

Sample derivation:

\[
\begin{align*}
\sigma \models \Box \varphi = \neg \Diamond \neg \varphi \quad \text{iff} \quad \neg \exists j \geq 0. \sigma[j \ldots] \not\models \varphi \\
\neg \exists j \geq 0. \sigma[j \ldots] \not\models \varphi \quad \text{iff} \quad \forall j \geq 0. \sigma[j \ldots] \models \varphi.
\end{align*}
\]

\( \Box \) means \( \forall i \geq 0. \exists j \geq i \).

\( \Diamond \) means \( \exists i \geq 0. \forall j \geq i \).
Semantics: when does a system satisfy an LTL spec?

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

- For infinite path fragment $π$ of $TS$, the satisfaction relation is defined by
  $$π \models φ \iff \text{trace}(π) \models φ.$$  
- For state $s ∈ S$, the satisfaction relation $|=\ $ is defined by
  $$s \models φ \iff (∀π ∈ \text{Paths}(s). \ π \models φ).$$  
- $TS$ satisfies $φ$, denoted $TS \models φ$, if $\text{Traces}(TS) \subseteq \text{Words}(φ)$.

Putting together:

- [Bullet 3 above]
- [Definition of satisfaction for LT properties]
- [Definition of $\text{Words}(φ)$]
- [Bullet 2 above]
Example: traffic light

System description
• Focus on lights in one particular direction
• Light can be any of three colors: green, yellow, read
• Atomic propositions = light color

Ordering specifications
• Liveness: “traffic light is green infinitely often”
  □◊green
• Chronological ordering: “once red, the light cannot become green immediately”
  □(red → ¬ ○ green)
• More detailed: “once red, the light always becomes green eventually after being yellow for some time”
  □(red → (◊ green ∧ (¬ green U yellow)))
  □(red → ○ (red U (yellow ∧ ○ (yellow U green))))

Progress property
• Every request will eventually lead to a response
  □(request → ◊ response)
Example: autonomous navigation

Specify safe, allowable, required, or desired behavior of system and/or environment.

Traffic rules:
- No collision: $\square (\text{dist}(x, \text{Obs}) \geq X_{\text{safe}} \land \text{dist}(x, \text{Loc(Veh)}) \geq X_{\text{safe}})$
- Obey speed limits: $\square ((x \in \text{Reduced Speed Zone}) \rightarrow (v \leq v_{\text{reduced}}))$
- Stay in travel lane unless blocked
- Intersection precedence & merging, stop line, passing,...

Goals:
- Eventually visit the check point: $\lozenge (x = \text{ck}_p)$
- Every time check point is reached, eventually come to start: $\square ((x = \text{ck}_p) \rightarrow \lozenge (x = \text{start}))$

Environment assumptions:
- Each intersection is clear infinitely often: $\square \lozenge (\text{Intersection} = \text{empty})$
- Limited sensing range, detect obstacles before too late,...
Consider the following transition system with $AP = \{a, b\}$

Property 1: $TS \models [] a$?
- Yes, all states are labeled with $a$

Property 2: $TS \models X (a \land b)$?
- No: From $s_2$ or $s_3$, there are transitions for which $a \land b$ doesn’t hold

Property 3: $TS \models [] (!b \rightarrow [] (a \land !b))$?
- True

Property 4: $TS \models b \lor (a \land !b)$?
- False: $(s_1s_2)^\omega$
LTL formulae $\varphi_1, \varphi_2$ are equivalent, denoted $\varphi_1 \equiv \varphi_2$, if $\text{Words}(\varphi_1) = \text{Words}(\varphi_2)$.

### Equivalence of LTL formulas

#### Duality Law

- $\neg \diamondsuit \varphi \equiv \bigcirc \neg \varphi$
- $\neg \lozenge \varphi \equiv \square \neg \varphi$
- $\neg \square \varphi \equiv \lozenge \neg \varphi$

#### Idempotency Law

- $\lozenge \lozenge \varphi \equiv \lozenge \varphi$
- $\square \square \varphi \equiv \square \varphi$
- $\varphi \mathnormal{U} (\varphi \mathnormal{U} \psi) \equiv \varphi \mathnormal{U} \psi$
- $(\varphi \mathnormal{U} \psi) \mathnormal{U} \psi \equiv \varphi \mathnormal{U} \psi$

#### Absorption Law

- $\lozenge \square \lozenge \varphi \equiv \square \lozenge \varphi$
- $\square \lozenge \square \varphi \equiv \lozenge \square \varphi$

#### Expansion Law

- $\varphi \mathnormal{U} \psi \equiv \psi \lor (\varphi \land \bigcirc (\varphi \mathnormal{U} \psi))$
- $\lozenge \psi \equiv \psi \lor \bigcirc \lozenge \psi$
- $\square \psi \equiv \psi \land \bigcirc \square \psi$

#### Distributive Law

- $\bigcirc (\varphi \mathnormal{U} \psi) \equiv (\bigcirc \varphi) \mathnormal{U} (\bigcirc \psi)$
- $\lozenge (\varphi \lor \psi) \equiv \lozenge \varphi \lor \lozenge \psi$
- $\square (\varphi \land \psi) \equiv \square \varphi \land \square \psi$

### Non-identities

- $\lozenge (a \land b) \neq \lozenge a \land \lozenge b$
- $\square (a \lor b) \neq \square a \lor \square b$
Specifying timed properties for synchronous systems

For 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.:

$$\bigcirc^k \varphi \triangleq \underbrace{\bigcirc \bigcirc \ldots \bigcirc}_k \varphi \quad \text{“$\varphi$ holds after (exactly) $k$ time instants”}.$$  

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

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

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

$$\square \leq_k \varphi \quad = \quad \neg \Diamond \leq_k \neg \varphi \quad = \quad \neg \bigvee_{0 \leq i \leq k} \bigcirc^i \neg \varphi.$$
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 execution: each thread should receive some fraction of processes time
- To rule out unrealistic behavior

Examples:

- N processors sharing a service: ensure each processor gets access to the service
- In a distributed protocol, ensure that each agent communicates with its “neighbors” regularly (infinitely often)
- Autonomous car at an intersection: ensure the intersection clears or the lights turn green in the future

Two issues:

- Implementation: How do we implement our algorithms to insure that we get “fairness” in execution?
- Specification: How do we model fairness in a formal way to reason about program correctness?
Fairness properties & their LTL representation

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

### Unconditional fairness

“Every process gets its turn infinitely often.”

Unconditional fairness $\quad ufair = \Box \Diamond \Psi.$

### Strong fairness

“Every process that is enabled infinitely often gets its turn infinitely often.”

Strong fairness $\quad sfair = \Box \Diamond \Phi \rightarrow \Box \Diamond \Psi.$

### Weak fairness

“Every process that is continuously enabled from a certain time on gets its turn infinitely often.”

Weak fairness $\quad wfair = \Diamond \Box \Phi \rightarrow \Box \Diamond \Psi.$

An **LTL fairness assumption:**

$$
\text{fair} = ufair \land sfair \land wfair.
$$

**Rules of thumb**

- strong (or unconditional) fairness: useful for solving contentions
- weak fairness: sufficient for resolving the non-determinism due to interleaving (i.e., a possible option is not consistently ignored)
LTL → Nondeterministic Buchi automata

**Theorem.** There exists an algorithm that takes an LTL formula $\Phi$ and returns a Büchi automaton $A$ such that

$$Words(\Phi) = \mathcal{L}_{\omega}(A)$$

A tool for constructing Buchi automata from LTL formulas: LTL2BA

[http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/index.php]
Branching time and computation tree logic (CTL)

LTL formulas are interpreted over paths; hence, there is a clear (and linear) notion of ordering of events over time.

Interpretation an LTL formula at a state: all paths starting from the state satisfy the formula.

LTL does not allow complicated quantification over the paths.
• E.g., “For every execution it is always possible to return to the initial state” cannot be specified in LTL.

Computation tree logic (CTL) allows evaluation over some or all paths.

\[
\forall \Box \exists \diamond \text{start} \quad \text{for all executions}
\]

\[
\forall \exists \forall \text{at any state} \quad \text{it is possible}
\]

\[
\exists \forall \Box \text{to eventually reach start}
\]
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 in 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>
<th>Hold/Don't Hold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possibly the system never goes down</td>
<td>$\exists \square \neg \text{down}$</td>
<td>Holds</td>
</tr>
<tr>
<td>Invariantly the system never goes down</td>
<td>$\forall \square \neg \text{down}$</td>
<td>Doesn't hold</td>
</tr>
<tr>
<td>It is always possible to start as new</td>
<td>$\forall \square \exists \Diamond \text{up}_3$</td>
<td>Holds</td>
</tr>
<tr>
<td>The system always eventually goes down and is operational until going down</td>
<td>$\forall ((\text{up}_3 \lor \text{up}_2) \lor \text{down})$</td>
<td>Doesn't hold</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

\[ \Phi ::= \text{true} \mid a \mid \Phi_1 \land \Phi_2 \mid \neg \Phi \mid \exists \varphi \]
\[ \varphi ::= \Phi \mid \varphi_1 \land \varphi_2 \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi_1 \cup \varphi_2 \]

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

\[ \forall \Box (\text{far} \rightarrow \forall \Diamond \leq 1 \forall \Box \leq 1 \text{ up}) \]
Summary: specifying behavior with (linear) temporal logic

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 (eg, infinitely often)

Example

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

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 (eg, TLC, Isabelle, PVS, SPIN, nuSMV, etc)