Computations are modeled using a state transition graph, also called a Kripke structure.

A Kripke structure $M = (P, S, I, L, R)$ consists of:

- Set of atomic propositions $P$
- States $S$
- Initial states $I \subseteq S$
- Labeling $L : S \mapsto 2^P$
- Transition relation $R \subseteq S \times S$

$P = \{p_0, p_1, p_2\}$, $S = \{s_0, s_1, s_2\}$

$I = \{s_0\}$

$L = \{(s_0, \{p_0, p_2\}), \ldots\}$

$R = \{(s_0, s_1), (s_1, s_2), \ldots\}$
Temporal Operators (Review)

There are five basic temporal operators we’ll use

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X p$</td>
<td>$p$ holds at the <em>next</em> point in time</td>
</tr>
<tr>
<td>$F p$</td>
<td>$p$ holds at <em>some future</em> point in time</td>
</tr>
<tr>
<td>$G p$</td>
<td>$p$ holds at <em>every point</em> in time</td>
</tr>
<tr>
<td>$p U q$</td>
<td>$p$ holds <em>until</em> $q$ holds</td>
</tr>
<tr>
<td>$p R q$</td>
<td>$p$ <em>releases</em> $q$: $q$ holds until $p$ (if it ever does)</td>
</tr>
</tbody>
</table>

These operators describe properties of a path $\pi$
Linear Temporal Logic (Review)

These operators allow us to define **linear temporal logic** (LTL)

LTL contains **state formulas** and **path formulas**

### State Formula

The syntax of state formulas is given by:

\[ f ::= \top | \bot | p | \neg f | f_1 \lor f_2 | f_1 \land f_2 \]

State formulas correspond to facts that hold in a particular state.

### Path Formula

An LTL formula is composed of the following elements:

\[ g ::= f | \neg g | g_1 \lor g_2 | g_1 \land g_2 | \mathbf{X} g | \mathbf{F} g | \mathbf{G} g | g_1 \mathbf{U} g_2 | g_1 \mathbf{R} g_s \]

Path formulas are evaluated along a particular path.
We mentioned that LTL universally quantifies over paths. It is also possible to have *existential* path quantification.

We extend state formulas with two new forms:

- \( \mathbf{E} \, f \) is now a state formula, where:
  \[
  M, s \models \mathbf{E} \ g \iff \text{there exists a path } \pi \text{ starting at } s \text{ where } M, \pi \models g
  \]

- \( \mathbf{A} \, f \) is now a state formula, where:
  \[
  M, s \models \mathbf{A} \ g \iff \text{for all paths } \pi \text{ starting at } s, M, \pi \models g
  \]

This defines the logic CTL*. LTL is a fragment of CTL*, where:

- Each formula implicitly starts with \( \mathbf{A} \)
- State formulas don’t have path quantifiers
LTL formulas describe linear-time properties: single paths

CTL* formulas describe branching-time properties

Path quantifiers can talk about multiple possible futures

For example, consider $\text{EF } p$
CTL is a fragment of CTL* where each temporal operator is preceded by a path quantifier

- If $f, f_1, f_2$ are state formulas, then $X f$, $F f$, $G f$, $f_1 U f_2$, and $f_1 R f_2$ are path formulas
- Importantly: path formulas are built directly from state formulas

### CTL Syntax

CTL formulas $\phi$ are built from the following grammar:

\[
\phi ::= \top | \bot | p \in P | \neg \phi | \phi_1 \lor \phi_2 | \phi_1 \land \phi_2 \\
| \text{EX } \phi | \text{EF } \phi | \text{EG } \phi | E [\phi_1 U \phi_2] | E [\phi_1 R \phi_2] \\
| \text{AX } \phi | \text{AF } \phi | \text{AG } \phi | A [\phi_1 U \phi_2] | A [\phi_1 R \phi_2]
\]
Example: Visualizing CTL ($\textbf{AX} \ p$)

What does $\textbf{AX} \ p$ look like?
Example: Visualizing CTL ($\text{EG } p$)

What does $\text{EG } p$ look like?
What does $E [p \mathbf{U} q]$ look like?
Example: Visualizing CTL ($A [p U q]$)

What does $A [p U q]$ look like?
What do the following formulas mean?

- **$\text{EF} (\text{start} \land \neg\text{ready})$**
  
  Can get to a state where $\text{start}$ holds by $\text{ready}$ doesn’t

- **$\text{AG} (\text{EF} \text{ restart})$**
  
  It is always possible (from any state) to enter $\text{restart}$

- **$\text{AF} (\text{AG} \ p)$**
  
  No matter what, $p$ eventually always holds on all paths

- **$\text{AG} (\text{AF} \ \text{crit})$**
  
  On all paths at all times, enter $\text{crit}$ infinitely often  
  Or, $\text{crit}$ is infinitely often true on all paths
Consider the last example from the previous slide:
\[ \text{AF (AG } p \text{)} \]

Couldn’t we just write \( \text{F G } p \)? Consider the following:

Does this satisfy \( \text{F G } p \)? Yes. What about \( \text{AF (AG } p \text{)} \)? No.
Consider one of the examples from the previous slide:

\[ \text{AG (EF } \text{restart}) \]

“It is always possible to enter \text{restart}.”

Let’s write this as an LTL formula

What about \text{G F } \text{restart}?

This says, “At all times, \text{restart} will eventually be entered”.

It’s too strong: we just need the possibility of returning to \text{restart}

This formula can’t be expressed in LTL
CTL and LTL are **incomparable** in terms of expressiveness.

There are formulas in both logics that can’t be expressed in the other.

We’ve seen an example CTL formula with no LTL equivalent:

\((\text{AG } (\text{EF } p))\)

We’ve also seen an LTL formula with no equivalence in CTL:

\((\text{F G } p)\)

These differences are not merely academic: both types of formulas are useful in practice.

- **AG** \((\text{EF } p)\) says that it is possible to reach a state with \(p\) regardless of the current state.
- **F G** \(p\) says that a state invariant will hold in the future.
Useful Temporal Properties

Typically, the properties we want to check fall into a few categories

- **Safety**: Properties which require that “nothing bad should ever happen”
- **Liveness**: Require that “something good always happens” in the future
- **Fairness**: Precludes “degenerate” infinite behaviors

These are inherently linear-time properties

Can often be approximated by branching properties
An important class of safety properties describes **invariant** behavior

Invariants state that some property over states holds at all times

- Mutual exclusion
- Deadlock freedom
- Well-formedness of data structures

If $\phi$ is a propositional formula over $P$, then the LTL formula $\mathbf{G} \phi$ is an invariant property
Consider two processes $P_1, P_2$ of the form:

```c
while (true) {
    :
    (noncritical actions)
    :
    while($y = 0$) skip;
    $y := y + 1$;
    critical section
    $y := y - 1$;
}
```

The variable $y$ is shared between $P_1$ and $P_2$

We’ll model each process using the predicates:

- $c_i$, that $P_i$ is in its critical section
- $n_i$, that $P_i$ is in non-critical section
- $w_i$, that $P_i$ is waiting for $y = 0$
- $y = 0, y = 1$
Example: Mutual Exclusion

The variable \( y \) is shared between \( P_1 \) and \( P_2 \)

We’ll model each process using the predicates:

- \( c_i \), that \( P_i \) is in its critical section
- \( n_i \), that \( P_i \) is in non-critical section
- \( w_i \), that \( P_i \) is waiting for \( y = 0 \)
- \( y = 0, y = 1 \)

Start by writing the TS for a single process
We’ll model the concurrent execution of $P_1, P_2$

We’ll create a **global** transition system

- Models state of both processes
- $P = \{c_1, c_2, n_1, n_2, w_1, w_2, y = 0, y = 1\}$
- Notice: shared state for $y = 0, y = 1$ is not duplicated
- Transitions account for all **interleavings**
- Intuitively, these are the outcome of nondeterministic choices between processes’ steps
Example: Mutual Exclusion

\[ n_1, n_1, y = 1 \]

\[ w_1, n_2, y = 1 \]
\[ n_1, w_2, y = 1 \]

\[ c_1, n_2, y = 0 \]

\[ w_1, w_2, y = 1 \]

\[ n_1, c_2, y = 0 \]

\[ c_1, w_2, y = 0 \]

\[ w_1, c_2, y = 0 \]
Recall that mutual exclusion is an invariant

\[ G \phi \]

What is \( \phi \)? \( P_1 \) and \( P_2 \) can’t be simultaneously critical

\[ \phi \iff \neg c_1 \lor \neg c_2 \]

Does the transition system satisfy this property?

**Yes.** Easy to see by visual inspection in this case

Algorithmically, invariant checking is a reachability problem
Violating an invariant is one type of “bad thing” that could happen. Generally, we’re interested in more than simple reachability.

General safety properties are defined in terms of bad path prefixes:

- Let $\phi$ be a property over paths.
- Let $\hat{\sigma}$ be a prefix such that no path $\sigma = \hat{\sigma} \cdots$ satisfies $\phi$.
- Note: we require $\hat{\sigma}$ to be finite.

$\phi$ is a safety property iff every path that violates $\phi$ has a bad prefix.

Think of a bad prefix as a witness of a violating instance.
Bad prefixes: example

The “bad prefix” definition shows that invariants are safety properties.

What are the bad prefixes of an invariant property?

$$G \phi$$

Any finite sequence ending with $$\neg \phi$$:

<table>
<thead>
<tr>
<th>$$\phi$$</th>
<th>$$\phi$$</th>
<th>$$\phi$$</th>
<th>$$\phi$$</th>
<th>$$\neg \phi$$</th>
</tr>
</thead>
</table>

What about in the mutual exclusion case?

$$\neg c_1 \lor \neg c_2$$ $$\neg c_1 \lor \neg c_2$$ $$\neg c_1 \lor \neg c_2$$ $$\neg c_1 \lor \neg c_2$$ $$c_1 \land c_2$$
We want to model a property of a standard traffic light:

“a red light must be preceded immediately by a yellow”

What are the atomic propositions?

$$P = \{\text{red}, \text{yellow}, \text{green}\}$$

What is an LTL formula?

$$G ((\text{yellow} \rightarrow X \text{red}) \land ((\text{green} \lor \text{red}) \rightarrow \neg X \text{red})))$$

Is this an invariant? No.

What is a bad prefix for this property?

\[
\begin{array}{cccccc}
\text{yellow} & \rightarrow & \text{red} & \rightarrow & \text{green} & \rightarrow \text{red} & \rightarrow \text{green} \\
\end{array}
\]
Safety properties prevent bad things from happening

It’s easy to design systems that don’t do bad things: do nothing

Usually this isn’t what we want, and liveness properties address this

Intuitively, “something good” will always happen in the future

This intuition reveals a key distinction from safety properties:

▶ Safety properties are violated in finite time
▶ Liveness properties can only be violated in infinite time
Liveness properties are defined in terms of **infinite path extensions**

- Let $\phi$ be a property over paths
- Let $\hat{\sigma}$ be a finite path prefix
- $\sigma$ is an infinite path extension if $\hat{\sigma}\sigma$ satisfies $\phi$

$\phi$ is a liveness property iff every finite prefix has an infinite extension

Put differently: a liveness property does not rule out any finite prefix

This implies that there are no finite witnesses for liveness
Think of the mutual exclusion example

Desirable liveness property: each $P_i$ enters $critical_i$ infinitely often

$$G F c_1 \land G F c_2$$

Another: each waiting process eventually enters its $critical_i$

$$G (w_i \rightarrow F c_i)$$

Any finite prefix can always be extended to satisfy these properties
Liveness: “Repeated Eventually”

Does this satisfy $\mathsf{G}\mathsf{F}c_1 \land \mathsf{G}\mathsf{F}c_2$? No. What’s a counterexample?
Does this satisfy $G(w_i \rightarrow F c_i)$? **No.** What’s a counterexample?
Fairness

Liveness violated because processes aren’t scheduled **fairly**

Unfairness arises from nondeterminism
- Choices that are globally biased against certain options
- Often these choices are unrealistic in practice
- Dealt with in the modelling, rather than checking, phase

To prove liveness, we want to rule out unfair paths from the model

**Fairness constraints** allow us to do this
There are several different types of fairness

- **Unconditional fairness**: Every process is executed infinitely often.
  
  \[ \text{for all } i, \ G F \ P_i \]

- **Strong fairness**: Every process that is enabled infinitely often gets its turn infinitely often when it is enabled.
  
  \[ \text{for all } i, (G F enabled_i) \rightarrow (G F P_i) \]

- **Weak fairness**: Every process that is at some point continuously enabled gets its turn infinitely often.
  
  \[ \text{for all } i, (F G enabled_i) \rightarrow (G F P_i) \]

“Enabled” means “able to execute” a particular transition.
Example: Weak Fairness

Consider the mutual exclusion example

A process is **enabled** if it is able to enter its critical section

Does this satisfy $G \left( w_i \rightarrow F c_i \right)$ under weak fairness?
Example: Weak Fairness

This doesn’t satisfy starvation freedom under weak fairness

Whenever $P_2$ enters critical, $P_1$ is not enabled

The antecedent that $P_i$ be enabled continuously is false
Example: Strong Fairness

Does this satisfy \( G (w_i \rightarrow F c_i) \) under strong fairness?

Yes, both processes enabled infinitely often along all paths

So a strongly-fair scheduler lets both execute infinitely often
Example: Unconditional Fairness

Does starvation freedom hold if the scheduler is unconditionally fair?

Unconditional Fairness: for all $i$, $G F P_i$

**Yes.** It held for strong fairness, which was:

Strong Fairness: for all $i$, $(G F enabled_i) \rightarrow (G F P_i)$

So we see that unconditional fairness implies strong fairness

In fact, we can say that:

Unconditional Fairness $\implies$ Strong Fairness $\implies$ Weak Fairness
▶ Continue discussing model checking
▶ Model checking techniques
▶ Fifth assignment goes out today
▶ Fourth assignment due tonight at 11pm