Model Checker: A program that checks if a (transition) system satisfies a (temporal) property.
High level definition

• **Model checking** is an automated technique that exhaustively explores the **state space** of a system, typically to see if an error state is **reachable**. It produces concrete **counter-examples**.
  - The **state explosion problem** refers to the large number of states in the model.
  - **Temporal logic** allows you to specify properties with concepts like “eventually” and “always”.
Explicit-state Temporal Logic Model Checking

- **Domain**: Continuously operating concurrent systems (e.g. operating systems, hardware controllers and network protocols)

- **Ongoing, reactive semantics**
  - Non-terminating, infinite computations
  - Manifest non-determinism

- **Systems are modeled by finite state machines**

- **Properties** are written in propositional temporal logic [Pnueli 77]

- **Verification procedure** is an exhaustive search of the state space of the design

- **Produces diagnostic counterexamples.**
Motivation: What can be Verified?

• Architecture
  - Will these two components interact properly?
    ▪ Allen and Garlan: Wright system checks architectures for deadlock

• Code
  - Am I using this component correctly?
    ▪ Microsoft’s Static Driver Verifier ensures complex device driver rules are followed
      ▪ Substantially reduced Windows blue screens
  - Is my code safe
    ▪ Will it avoid error conditions?
    ▪ Will it be responsive, eventually accepting the next input?

• Security
  - Is the protocol I’m using secure
    ▪ Model checking has found defects in security protocols
Temporal Properties

- **Temporal Property**: A property with time-related operators such as “invariant” or “eventually”
- **Invariant($p$)**: is true in a state if property $p$ is true in every state on all execution paths starting at that state
  - The Invariant operator has different names in different temporal logics:
    - $G$, $AG$, $\Box$ (“goal” or “box” or “forall”)
- **Eventually($p$)**: is true in a state if property $p$ is true at some state on every execution path starting from that state
  - $F$, $AF$, $\Diamond$ (“diamond” or “future” or “exists”)

invariant: is true in a state if property $p$ is true in every state on all execution paths starting at that state. The Invariant operator has different names in different temporal logics: $G$, $AG$, $\Box$ (“goal” or “box” or “forall”). Eventually($p$): is true in a state if property $p$ is true at some state on every execution path starting from that state. $F$, $AF$, $\Diamond$ (“diamond” or “future” or “exists”).
What is Model Checking?

Does model M satisfy a property P?
(written $M \models P$

What is “M”?

What is “P”?

What is “satisfy”?
What is “M”?  

Example Program:  

**precondition: numTickets > 0**  
reserved = false;  
while (true) {  
    getQuery();  
    if (numTickets > 0 && !reserved)  
        reserved = true;  
    if (numTickets > 0 && reserved) {  
        reserved = false;  
        numTickets--;  
    }  
}
What is “M”?  

Example Program:

`precondition: numTickets > 0`
reserved = false;
while (true) {
    getQuery();
    if (numTickets > 0 && !reserved)
        reserved = true;
    if (numTickets > 0 && reserved) {
        reserved = false;
        numTickets--;
    }
}

What is interesting about this?
Are tickets available?    a
Is a ticket reserved?    r

```
<table>
<thead>
<tr>
<th>State</th>
<th>Tickets</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>nT=2, r=false</td>
<td></td>
<td>true</td>
</tr>
<tr>
<td>nT=2, r=true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nT=1, r=false</td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>nT=1, r=true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nT=0, r=false</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

9
What is “M”?

Abstracted Program: fewer states

**precondition: available == true**

```java
reserved = false;
while (true) {
    getQuery();
    if (available && !reserved)
        reserved = true;
    if (available && reserved) {
        reserved = false;
        available = ?;
    }
}
```

State Transition Graph or Kripke Model
What is “M”? 

Abstracted Program: fewer states

precondition: available == true
reserved = false;
while (true) {
    getQuery();
    if (available && !reserved)
        reserved = true;
    if (available && reserved) {
        reserved = false;
        available = ?;
    }
}
What is “M”? 

**State:** valuations to all variables
- concrete state: (numTickets=5, reserved=false)
- abstract state: (a=true, r=false)

**Initial states:** subset of states

**Arcs:** transitions between states

**Atomic Propositions:**
- a: numTickets > 0
- r: reserved = true

State Transition Graph or Kripke Model
An Example Concurrent Program

- A simple **concurrent mutual exclusion program**
- Two processes execute asynchronously
- There is a shared variable **turn**
- Two processes use the shared variable to ensure that they are **not in the critical section at the same time**
- Can be viewed as a “fundamental” program: any bigger concurrent one would include this one

10: while True do
11:    wait(turn = 0);
       // critical section
12:    work(); turn := 1;
13:  end while;

|| // concurrently with

20: while True do
21:   wait(turn = 1);
       // critical section
22:   work(); turn := 0;
23: end while
Reachable States of the Example Program

Each state is a valuation of all the variables: turn and the two program counters for two processes.

Next: formalize this intuition …
What is “M”? A Labelled Transition System

\[ M = \langle S, S_0, R, L \rangle \]

Kripke structure:
- \( S \) – finite set of states
What is “M”? A Labelled Transition System

\[ M = \langle S, S_0, R, L \rangle \]

Kripke structure:
- \( S \) – finite set of states
- \( S_0 \subseteq S \) – set of initial states
What is “M”? A Labelled Transition System

\[ M = \langle S, S_0, R, L \rangle \]

Kripke structure:
- \( S \) – finite set of states
- \( S_0 \subseteq S \) – set of initial states
- \( R \subseteq S \times S \) – set of arcs
What is “M”? A Labelled Transition System

\[ M = \langle S, S_0, R, L \rangle \]

Kripke structure:
- \( S \) – finite set of states
- \( S_0 \subseteq S \) – set of initial states
- \( R \subseteq S \times S \) – set of arcs
- \( L : S \rightarrow 2^{\text{AP}} \) – mapping from states to a set of atomic propositions

(e.g., “x=5 ∈ AP)
- Atomic propositions capture basic properties
- For software, atomic props depend on variable values
- The labeling function labels each state with the set of propositions true in that state
Atomic Propositions

• We must decide in advance which facts are important.
  • E.g. “x=5” or “x=6”, also relations like “x<y”

• Example: “In all the reachable states (configurations) of the system, the two processes are never in the critical section at the same time”
  • Equivalently, we can say that: Invariant(¬(pc1=12 ∧ pc2=22))

• Also: “Eventually the first process enters the critical section”
  • Eventually(pc1=12)

• “pc1=12”, “pc2=22” are atomic properties
Model of Computation

State Transition Graph

Unwind State Graph to obtain traces. A trace is an infinite sequence of states. The semantics of a FSM is a set of traces.
Model of Computation

State Transition Graph

Unwind State Graph to obtain traces. A *trace* is an infinite sequence of states. The *semantics* of a FSM is a set of traces.
Unwind State Graph to obtain traces. A trace is an infinite sequence of states. The semantics of a FSM is a set of traces.
Model of Computation

Unwind State Graph to obtain traces. A *trace* is an infinite sequence of states. The *semantics* of a FSM is a set of traces.
Model of Computation

State Transition Graph

Computation Traces

Unwind State Graph to obtain traces. A *trace* is an infinite sequence of states. The *semantics* of a FSM is a set of traces.
Model of Computation

State Transition Graph

Computation Traces

Unwind State Graph to obtain traces. A trace is an infinite sequence of states. The semantics of a FSM is a set of traces.
Model of Computation

Represent all traces with an infinite computation tree
What is “P”? 

Different kinds of temporal logics 

**Syntax:** What are the formulas in the logic? 

**Semantics:** What does it mean for model $M$ to satisfy formula $P$? 

**Formulas:** 
- Atomic propositions: properties of states 
- Temporal Logic Specifications: properties of traces.
Examples:  

**Safety** (mutual exclusion): no two processes can be at a critical section at the same time

**Liveness** (absence of starvation): every request will be eventually granted

Temporal logics differ according to how they handle branching in the underlying computation tree.

In a linear temporal logic (LTL), operators are provided for describing system behavior along a single computation path.

In a branching-time logic (CTL), the temporal operators quantify over the paths that are possible from a given state.
Temporal Logics

• There are four basic temporal operators:
  • $X p = \text{Next } p$, $p$ holds in the next state
  • $G p = \text{Globally } p$, $p$ holds in every state, $p$ is an invariant
  • $F p = \text{Future } p$, $p$ will hold in a future state, $p$ holds eventually
  • $p U q = p \text{ Until } q$, assertion $p$ will hold until $q$ holds

• Precise meaning of these temporal operators are defined on execution paths
Execution Paths

• A path $\pi$ in $M$ is an infinite sequence of states $(s_0, s_1, s_2, ...)$, such that $\forall i \geq 0. (s_i, s_{i+1}) \in R$
  o $\pi^i$ denotes the suffix of $\pi$ starting at $s_i$

• $M, \pi \models f$ means that $f$ holds along path $\pi$ in the Kripke structure $M$,
  o “the path $\pi$ in the transition system makes the temporal logic predicate $f$ true”
  o Example: $M, \pi \models \mathbf{G} (\neg (pc1=12 \land pc2=22))$

• In some temporal logics one can quantify the paths starting from a state using path quantifiers
  o $A :$ for all paths
  o $E :$ there exists a path
Summary: Formulas over States and Paths

• State formulas
  o Describe a property of a state in a model M
  o If \( p \in AP \), then \( p \) is a state formula
  o If \( f \) and \( g \) are state formulas, then \( \neg f, f \land g \) and \( f \lor g \) are state formulas
  o If \( f \) is a path formula, then \( \mathbf{E} f \) and \( \mathbf{A} f \) are state formulas

• Path formulas
  o Describe a property of an infinite path through a model M
  o If \( f \) is a state formula, then \( f \) is also a path formula
  o If \( f \) and \( g \) are path formulas, then \( \neg f, f \land g, f \lor g, \mathbf{X} f, \mathbf{F} f, \mathbf{G} f, \) and \( f \mathbf{U} g \) are path formulas
LTL logic operators wrt Paths

Linear Time Logic (LTL) [Pnueli 77]: logic of temporal sequences.

- LTL properties are constructed from atomic propositions in AP; logical operators $\land, \lor, \neg$; and temporal operators $X, G, F, U$.
- The semantics of LTL properties is defined on paths:
  - $\alpha$: $\alpha$ holds in the current state (atomic)
  - $X\alpha$: $\alpha$ holds in the next state (Next)
  - $F\gamma$: $\gamma$ holds eventually (Future)
  - $G\lambda$: $\lambda$ holds from now on (Globally)
  - $(\alpha U \beta)$: $\alpha$ holds until $\beta$ holds (Until)
Satisfying Linear Time Logic

• Given a transition system $T = (S, I, R, L)$ and an LTL property $p$, $T$ satisfies $p$ if all paths starting from all initial states $I$ satisfy $p$.

• Example LTL formulas:
  - **Invariant**($\neg (pc1=12 \land pc2=22)$):
    $$G(\neg (pc1=12 \land pc2=22))$$
  - **Eventually**($pc1=12$):
    $$F(pc1=12)$$
- **Invariant**($\neg (pc1=12 \land pc2=22)$):
  \[ G(\neg (pc1=12 \land pc2=22)) \]
- **Eventually**($pc1=12$):
  \[ F(pc1=12) \]
LTL Satisfiability Examples

On this path: F p holds, G p does not hold, p does not hold, X p does not hold, X (X p) holds, X (X (X p)) does not hold

On this path: F p holds, G p holds, p holds, X p holds, X (X p) holds, X (X (X p)) holds
Typical LTL Formulas

- \( G (Req \implies F \text{Ack}) \): whenever Request occurs, it will be eventually Acknowledged.

- \( G (DeviceEnabled) \): DeviceEnabled always holds on every computation path.

- \( G (F \text{Restart}) \): Fairness: from any state one will eventually get to a Restart state. I.e. Restart states occur infinitely often.

- \( G (Reset \implies F \text{Restart}) \): whenever the reset button is pressed one will eventually get to the Restart state.

- Pedantic note:
  - G is sometimes written \( \Box \)
  - F is sometimes written \( \lozenge \)
Practice Writing Properties

• If the door is locked, it will not open until someone unlocks it
  o assume atomic predicates locked, unlocked, open

• If you press ctrl-C, you will get a command line prompt

• The saw will not run unless the safety guard is engaged
Practice Writing Properties

• If the door is locked, it will not open until someone unlocks it
  o assume atomic predicates locked, unlocked, open
  o \( G (\text{locked} \implies (\neg \text{open U unlocked})) \)

• If you press ctrl-C, you will get a command line prompt
  o \( G (\text{ctrlC} \implies F \text{ prompt}) \)

• The saw will not run unless the safety guard is engaged
  o \( G (\neg \text{safety} \implies \neg \text{running}) \)
LTL Model Checking Example

- Pressing Start will eventually result in heat
- $G(\text{Start} \Rightarrow F \text{Heat})$
LTL Model Checking

• \( f \) (primitive formula)
  o Just check the properties of the current state

• \( Xf \)
  o Verify \( f \) holds in all successors of the current state

• \( Gf \)
  o Find all reachable states from the current state, and ensure \( f \) holds in all of them
    ▪ use depth-first or breadth-first search

• \( f \cup g \)
  o Do a depth-first search from the current state. Stop when you get to a \( g \) or you loop back on an already visited state. Signal an error if you hit a state where \( f \) is false before you stop.

• \( Ff \)
  o Harder. Intuition: look for a path from the current state that loops back on itself, such that \( f \) is false on every state in the path. If no such path is found, the formula is true.
    ▪ Reality: use Büchi automata
LTL Model Checking Example

- Pressing Start will eventually result in heat
- $\neg \text{Start} \implies \neg \text{F Heat}$
LTL Model Checking Example

- **You try:** The oven doesn’t heat up until the door is closed.

\( \neg (\neg \text{Heat}) \cup \text{Close} \)
\( \neg (\neg \text{Heat}) \mathcal{W} \text{Close} \)
\( G (\text{not Closed} \Rightarrow \neg \text{Heat}) \)
Semantics of LTL Formulas

\[ M, \pi \models p \iff \pi = s \ldots \land p \in L(s) \]

\[ M, \pi \models \neg g \iff M, \pi \not\models g \]

\[ M, \pi \models g_1 \land g_2 \iff M, \pi \models g_1 \land M, \pi \models g_2 \]

\[ M, \pi \models g_1 \lor g_2 \iff M, \pi \models g_1 \lor M, \pi \models g_2 \]
Semantics of LTL Formulas

\[ M, \pi \models p \iff \pi = s \ldots \land p \in L(s) \]

\[ M, \pi \models \neg g \iff M, \pi \not\models g \]

\[ M, \pi \models g_1 \land g_2 \iff M, \pi \models g_1 \land M, \pi \models g_2 \]

\[ M, \pi \models g_1 \lor g_2 \iff M, \pi \models g_1 \lor M, \pi \models g_2 \]

\[ M, \pi \models X g \iff M, \pi^1 \models g \]
Semantics of LTL Formulas

\[ M, \pi \models p \iff \pi = s \ldots \land p \in L(s) \]

\[ M, \pi \models \neg g \iff M, \pi \not\models g \]

\[ M, \pi \models g_1 \land g_2 \iff M, \pi \models g_1 \land M, \pi \models g_2 \]

\[ M, \pi \models g_1 \lor g_2 \iff M, \pi \models g_1 \lor M, \pi \models g_2 \]

\[ M, \pi \models X g \iff M, \pi^1 \models g \]

\[ M, \pi \models F g \iff \exists k \geq 0 \mid M, \pi^k \models g \]
Semantics of LTL Formulas

\[ M, \pi \models p \iff \pi = s \ldots \land p \in L(s) \]

\[ M, \pi \models \neg g \iff M, \pi \not\models g \]

\[ M, \pi \models g_1 \land g_2 \iff M, \pi \models g_1 \land M, \pi \models g_2 \]

\[ M, \pi \models g_1 \lor g_2 \iff M, \pi \models g_1 \lor M, \pi \models g_2 \]

\[ M, \pi \models X g \iff M, \pi^1 \models g \]

\[ M, \pi \models F g \iff \exists k \geq 0 \mid M, \pi^k \models g \]

\[ M, \pi \models G g \iff \forall k \geq 0 \mid M, \pi^k \models g \]
Semantics of LTL Formulas

\[
\begin{align*}
M, \pi \models p & \iff \pi = s... \land p \in L(s) \\
M, \pi \models \neg g & \iff M, \pi \not\models g \\
M, \pi \models g_1 \land g_2 & \iff M, \pi \models g_1 \land M, \pi \models g_2 \\
M, \pi \models g_1 \lor g_2 & \iff M, \pi \models g_1 \lor M, \pi \models g_2 \\
M, \pi \models Xg & \iff M, \pi I \models g \\
M, \pi \models Fg & \iff \exists k \geq 0 \mid M, \pi^k \models g \\
M, \pi \models Gg & \iff \forall k \geq 0 \mid M, \pi^k \models g \\
M, \pi \models g_1 \mathcal{U} g_2 & \iff \exists k \geq 0 \mid M, \pi^k \models g_2 \\
& \quad \land \forall 0 \leq j < k M, \pi^j \models g_1
\end{align*}
\]
Semantics of LTL Formulas

\[ M, \pi \models p \iff \pi = s \ldots \land p \in L(s) \]

\[ M, \pi \models \neg g \iff M, \pi \not\models g \]

\[ M, \pi \models g_1 \land g_2 \iff M, \pi \models g_1 \land M, \pi \models g_2 \]

\[ M, \pi \models g_1 \lor g_2 \iff M, \pi \models g_1 \lor M, \pi \models g_2 \]

\[ M, \pi \models X g \iff M, \pi^1 \models g \]

\[ M, \pi \models F g \iff \exists k \geq 0 \mid M, \pi^k \models g \]

\[ M, \pi \models G g \iff \forall k \geq 0 \mid M, \pi^k \models g \]

\[ M, \pi \models g_1 \mathcal{U} g_2 \iff \exists k \geq 0 \mid M, \pi^k \models g_2 \land \forall 0 \leq j < k M, \pi^j \models g_1 \]

\[ g_2 \] must eventually hold

semantics of “until” in English are potentially unclear—that’s why we have a formal definition
Semantics of Formulas

\[ M, s \models p \iff p \in \mathcal{L}(s) \]

\[ M, s \models \neg f \iff M, s \not\models f \]

\[ M, s \models f_1 \land f_2 \iff M, s \models f_1 \land M, s \models f_2 \]

\[ M, s \models \forall \pi = s \ldots \lor M, \pi \models g_1 \]

\[ M, s \models \exists \pi = s \ldots \lor M, \pi \models g_1 \]

\[ M, \pi \models f \iff \pi = s \ldots \land M, s \models f \]

\[ M, \pi \models \neg g \iff M, \pi \not\models g \]

\[ M, \pi \models g_1 \land g_2 \iff M, \pi \models g_1 \land M, \pi \models g_2 \]

\[ M, \pi \models g_1 \lor g_2 \iff M, \pi \models g_1 \lor M, \pi \models g_2 \]

\[ M, \pi \models X g \iff M, \pi^1 \models g \]

\[ M, \pi \models F g \iff \exists k \geq 0 \mid M, \pi^k \models g \]

\[ M, \pi \models G g \iff \forall k \geq 0 \mid M, \pi^k \models g \]

\[ M, \pi \models g_1 \cup g_2 \iff \exists k \geq 0 \mid M, \pi^k \models g_2 \]

\[ \land 0 \leq j < k \mid M, \pi^j \models g_1 \]
Model Checking Complexity

- Given a transition system $T = (S, I, R, L)$ and an LTL formula $f$
  - One can check if the transition system satisfies the temporal logic formula $f$ in $O(2^{|f|} \times (|S| + |R|))$ time

- Given a transition system $T = (S, I, R, L)$ and a CTL formula $f$
  - One can check if a state of the transition system satisfies the temporal logic formula $f$ in $O(|f| \times (|S| + |R|))$ time

- Model checking procedures can generate counter-examples without increasing the complexity of verification (= “for free”)

State Space Explosion

Problem:
Size of the state graph can be exponential in size of the program (both in the number of the program variables and the number of program components or processes)

\[ M = M_1 \parallel \ldots \parallel M_n \]

If each \( M_i \) has just 2 local states, potentially \( 2^n \) global states

Research Directions: State space reduction
Explicit-State Model Checking

- One can show the complexity results using depth first search algorithms
  - The transition system is a directed graph
  - CTL model checking is multiple depth first searches (one for each temporal operator)
  - LTL model checking is one nested depth first search (i.e., two interleaved depth-first-searches)
- Such algorithms are called explicit-state model checking algorithms.
Temporal Properties \equiv Fixpoints

- States that satisfy $AG(p)$ are all the states which are not in $EF(\neg p)$ (= the states that can reach $\neg p$)
- Compute $EF(\neg p)$ as the **fixpoint** of $Func$: $2^S \rightarrow 2^S$
- Given $Z \subseteq S$,
  - $Func(Z) = \neg p \cup \text{reach-in-one-step}(Z)$
  - or $Func(Z) = \neg p \cup \text{EX}(Z)$

- Actually, $EF(\neg p)$ is the **least-fixpoint** of $Func$
  - smallest set $Z$ such that $Z = Func(Z)$
  - to compute the least fixpoint, start the iteration from $Z=\emptyset$, and apply the $Func$ until you reach a fixpoint
  - This can be computed (unlike most other fixpoints)
Pictorial Backward Fixpoint

Initial states

- initial states that violate $\text{AG}(p)$
  - initial states that satisfy $\text{EF}(\neg p)$
- states that can reach $\neg p = \text{EF}(\neg p)$
  - states that violate $\text{AG}(p)$

Inverse Image of $\neg p = \text{EX}(\neg p)$

This fixpoint computation can be used for:
- verification of $\text{EF}(\neg p)$
- or falsification of $\text{AG}(p)$

... and a similar forward fixpoint handles the other cases
Symbolic Model Checking

- Symbolic Model Checking represent state sets and the transition relation as *Boolean logic formulas*
  - Fixpoint computations manipulate *sets of states* rather than individual states
  - Recall: we needed to compute EX(Z), but Z ⊆ S
- Forward and backward fixpoints can be computed by iteratively manipulating these formulas
  - Forward, inverse image: Existential variable elimination
  - Conjunction (intersection), disjunction (union) and negation (set difference), and equivalence check
- Use an efficient data structure for manipulation of Boolean logic formulas: *Binary Decision Diagrams (BDDs)*
To produce the explicit counter-example, use the “onion-ring method”

- A counter-example is a valid execution path
- For each Image Ring (= set of states), find a state and link it with the concrete transition relation $R$
- Since each Ring is “reached in one step from previous ring” (e.g., $\text{Ring#3} = \text{EX(Ring#4)}$) this works
- Each state $z$ comes with $L(z)$ so you know what is true at each point (= what the values of variables are)
Model Checking Performance/Examples

• Performance:
  o Model Checkers today can routinely handle systems with between 100 and 300 state variables.
  o Systems with $10^{120}$ reachable states have been checked.
  o By using appropriate abstraction techniques, systems with an essentially unlimited number of states can be checked.

• Notable examples:
  o **IEEE Scalable Coherent Interface** – In 1992 Dill’s group at Stanford used Murphi to find several errors, ranging from uninitialized variables to subtle logical errors
  o **IEEE Futurebus** – In 1992 Clarke’s group at CMU found previously undetected design errors
  o **PowerScale multiprocessor** (processor, memory controller, and bus arbiter) was verified by Verimag researchers using CAESAR toolbox
  o **Lucent telecom**. protocols were verified by FormalCheck – errors leading to lost transitions were identified
  o **PowerPC 620 Microprocessor** was verified by Motorola’s Verdict model checker.
Efficient Algorithms for LTL Model Checking

• Use Büchi automata
  o Beyond the scope of this course

• Canonical reference on Model Checking:
Computation Tree Logics

- Formulas are constructed from *path quantifiers* and *temporal operators*:

**1. Path Quantifiers:**
- **A** – “for every path”
- **E** – “there exists a path”

**2. Temporal Operator:**
- **Xα** - α holds next time
- **Fα** - α holds sometime in the future
- **Gα** - α holds globally in the future
- **α Uβ** - α holds until β holds

*LTL: start with an A and then use only Temporal Operators*
The Logic CTL

In a branching-time logic (CTL), the temporal operators quantify over the paths that are possible from a given state \( s_0 \). Requires each temporal operator (\( X \), \( F \), \( G \), and \( U \)) to be preceded by a path quantifier (\( A \) or \( E \)).

\[
\begin{align*}
M, s_0 &\models AG \ c \\
M, s_0 &\models AF \ c \\
M, s_0 &\models EF \ c \\
M, s_0 &\models EG \ c
\end{align*}
\]
Remember the Example
Linear vs. Branching Time

One path starting at state (turn=0, pc1=10, pc2=20)

Linear Time View

Branching Time View

A computation tree starting at state (turn=0, pc1=10, pc2=20)
Example/Typical CTL Formulas

• **EF** \((\text{Started} \land \neg \text{Ready})\): it is possible to get to a state where \text{Started} holds but \text{Ready} does not hold.

• **AG** \((\text{Req} \Rightarrow \text{AF Ack})\): whenever \text{Request} occurs, it will be eventually \text{Acknowledged}.

• **AG** \((\text{DeviceEnabled})\): \text{DeviceEnabled} always holds on every computation path.

• **AG** \((\text{EF Restart})\): from any state it is possible to get to the \text{Restart} state.
At state $s$:
EF $p$, EX (EX $p$),
AF ($\neg p$), $\neg p$ holds
AF $p$, AG $p$,
AG ($\neg p$), EX $p$,
EG $p$, $p$ does not hold

At state $s$:
EF $p$, AF $p$,
EX (EX $p$),
EX $p$, EG $p$, $p$ holds
AG $p$, AG ($\neg p$),
AF ($\neg p$) does not hold

At state $s$:
EF $p$, AF $p$,
AG $p$, EG $p$,
Ex $p$, AX $p$, $p$ holds
EG ($\neg p$), EF ($\neg p$),
does not hold
Trivia

- AG(EF ρ) cannot be expressed in LTL
  - Reset property: from every state it is possible to get to ρ
    - But there might be paths where you never get to ρ
  - Different from A(GF ρ)
    - Along each possible path, for each state in the path, there is a future state where ρ holds
    - Counterexample: ababab...
Trivia

• A(FG p) cannot be expressed in CTL
  o Along all paths, one eventually reaches a point where p always holds from then on
    ▪ But at some points in some paths where p always holds, there might be a diverging path where p does not hold
  o Different from AF(AG p)
    ▪ Along each possible path there exists a state such that p always holds from then on
    ▪ Counterexample: the path that stays in s₀
Linear vs Branching-Time logics

- LTL is a linear time logic: when determining if a path satisfies an LTL formula we are only concerned with a single path
- CTL is a branching time logic: when determining if a state satisfies a CTL formula we are concerned with multiple paths
  - The computation is viewed as a tree which contains all the paths
  - The computation tree is obtained by unrolling the transition relation
- The expressive powers of CTL and LTL are incomparable ($LTL \subseteq CTL^*$, $CTL \subseteq CTL^*$)
  - Basic temporal properties can be expressed in both logics
  - Not in this lecture, sorry! (Take a class on Modal Logics)
Some advantages of LTL

- LTL properties are preserved under “abstraction”: i.e., if \( M \) “approximates” a more complex model \( M' \), by introducing more paths, then
  
  \[ M \models \psi \iff M' \models \psi \]

- “counterexamples” for LTL are simpler: single executions (not trees).

- The automata-theoretic approach to LTL model checking is simpler (no tree automata).

- Most properties people are interested in are (anecdottally) linear-time.

Some advantages of BT

- BT allows expression of some useful properties like ‘reset’.

- CTL, a limited fragment of the more complete BT logic CTL*, can be model checked in time linear in the formula size (as well as in the transition system).
  
  - But formulas are usually smaller than models, so this isn’t as important as it may first seem.

- Some BT logics, like \( \mu \)-calculus and CTL, are well-suited for the kind of fixed-point computation scheme used in symbolic model checking.
Software Model Checking?

- Use a finite state programming language, like executable design specifications (Statecharts, xUML, etc.).
- Extract finite state machines from programs written in conventional programming languages.
- Unroll the state machine obtained from the executable of the program.
- Use a combination of the state space reduction techniques to avoid generating too many states.
  - Verisoft (Bell Labs)
  - FormalCheck/xUML (UT Austin, Bell Labs)
  - ComFoRT (CMU/SEI)
- Use static analysis to extract a finite state skeleton from a program, model check the result.
  - Bandera – Kansas State
  - Java PathFinder – NASA Ames
  - SLAM/Bebop - Microsoft