Linear Time Logic (LTL)

- LTL properties are constructed from atomic propositions in AP: logical operators &, ∨, ¬ and temporal operators X, G, F, U.
- The semantics of LTL is defined on paths:
  Given a path h:
  \[ h \models p \quad \text{iff} \quad L(h_0, ap) \quad \text{atomic prop} \]
  \[ h \models X p \quad \text{iff} \quad h^1 \models p \quad \text{next} \]
  \[ h \models F p \quad \text{iff} \quad \exists i \geq 0. h^i \models p \quad \text{future} \]
  \[ h \models G p \quad \text{iff} \quad \forall i \geq 0. h^i \models p \quad \text{globally} \]
  \[ h \models p \quad \text{U} q \quad \text{iff} \quad \exists i \geq 0. h^i \models q \text{ and } \forall j < i. h^j \models p \quad \text{until} \]

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 \( p \).

Computation Tree Logic (CTL)

- In CTL temporal properties use path quantifiers: \( \forall \) : for all paths, \( \exists \) : there exists a path.
- The semantics of CTL is defined on states:
  Given a state \( s \):
  \[ s \models ap \quad \text{iff} \quad L(s, ap) \]
  \[ s_0 \models \exists x p \quad \text{iff} \quad \exists \text{ a path } (s_0, s_1, s_2, ...) \text{ s}_1 \models p \]
  \[ s_0 \models \forall x p \quad \text{iff} \quad \forall \text{ paths } (s_0, s_1, s_2, ...) \text{ s}_1 \models p \]
  \[ s_0 \models G p \quad \text{iff} \quad \forall i \geq 0. s_i \models p \]
  \[ s_0 \models A p \quad \text{iff} \quad \forall i \geq 0. s_i \models p \]
  ...

Linear vs. Branching Time

- 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.
  - In CTL the computation is instead viewed as a computation tree which contains all the paths.

The expressive powers of CTL and LTL are incomparable (LTL ⊆ CTL*, CTL ⊆ CTL*):
- Basic temporal properties can be expressed in both logics.
- Not in this lecture, sorry (Take a class on Modal Logics).
Recall the Example

This is a labeled transition system

Linear vs. Branching Time

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

LTL Satisfiability Examples

□p does not hold  ◯p holds

On this path:
Holds  Does Not Hold

LTL Satisfiability Examples

□p does not hold  ◯p holds

On this path:
Holds  Does Not Hold

LTL Satisfiability Examples

□p does not hold  ◯p holds

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

CTL Satisfiability Examples

□p does not hold  ◯p holds

At state s:
Holds  Does Not Hold
**CTL Satisfiability Examples**

- **p does not hold**

<table>
<thead>
<tr>
<th>s</th>
<th>Holds</th>
<th>Does Not Hold</th>
</tr>
</thead>
</table>

- **p holds**

At state s:
- EF p, EX (EX p), AF (¬¬¬¬ p), ¬¬¬¬ p holds
- AF p, AG p, AG (¬¬¬¬ p), EX p, EG p, p does not hold
- AG p, AG (¬¬¬¬ p), AF (¬¬¬¬ p) does not hold

**Model Checking Complexity**

- 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 formula \( f \) in \( O(|f| \times (|S| + |R|)) \) time
  - Multiple depth first searches (one for each temporal operator)
    - explicit-state model checking

**State Space Explosion**

- The complexity of model checking increases linearly with respect to the size of the transition system \( (|S| + |R|) \)
- However, the size of the transition system \( (|S| + |R|) \) is exponential in the number of variables and number of concurrent processes
- This exponential increase in the state space is called the **state space explosion**
  - Dealing with it is one of the major challenges in model checking research

**Algorithm:**

**Temporal Properties = Fixpoints**

- States that satisfy \( AG(p) \) are all the states which are not in \( EF(¬p) \) (the states that can reach \( ¬p \))
- Compute \( EF(¬p) \) as the **fixed point** of Func: \( 2^S \to 2^S \)
  - **Called the inverse image of Z**
- Given \( Z \subseteq S \)
  - Func(Z) = \( ¬p \cup \text{reach-in-one-step}(Z) \)
- Actually, \( EF(¬p) \) is the **least-fixed point** of Func
  - Smallest set \( Z \) such that \( Z = \text{Func}(Z) \)
  - to compute the least fixed point, start the iteration from \( Z=\emptyset \), and apply the Func until you reach a fixed point
  - This can be **computed** (unlike most other fixed points)
**Pictorial Backward Fixed Point**

Initial states

\[(\text{initial states that violate } \mathcal{AG}(p)) = (\text{initial states that satisfy } \mathcal{EF}(\neg p))\]

\[(\text{states that can reach } \neg p = \mathcal{EF}(\neg p)) = (\text{states that violate } \mathcal{AG}(p))\]

This fixed point computation can be used for:
- verification of \(\mathcal{EF}(\neg p)\)
- or falsification of \(\mathcal{AG}(p)\)

... and similar fixed points handle the other cases

**Symbolic Model Checking**

- **Symbolic model checking** represent state sets and the transition relation as Boolean logic formulas
  - Fixed point computations manipulate sets of states rather than individual states
  - Recall: we needed to compute reach-in-one-step (Z), but \(Z \subseteq S\)
- Fixed points can be computed by iteratively manipulating these formulas
- Use an efficient data structure for manipulation of Boolean logic formulas
  - Binary Decision Diagrams (BDDs)
- SMV (Symbolic Model Verifier) was the first CTL model checker to use BDDs

**Binary Decision Diagrams (BDDs)**

- Efficient representation for boolean functions (a set can be viewed as a function)
- Disjunction, conjunction complexity: at most quadratic
- Negation complexity: constant
- Equivalence checking complexity: constant or linear
- Image computation complexity: can be exponential

**Building Up To Software Model Checking via Counterexample Guided Abstraction Refinement**

There are easily dozens of papers.

We will skim.

**For Next Time**

- Post about today’s class and reading
- Read “Lazy Abstraction”
  - for the main ideas, ok to skim Sec. 7