Lecture 23: Model Checking and Temporal Logics

17-355/17-655/17-819: Program Analysis

Rohan Padhye and Jonathan Aldrich

April 26, 2021

With thanks for slides developed by Claire Le Goues, Natasha Sharygina, and Wes Weimer, used and adapted with permission.

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”.

[Clarke,Emerson 81][Queille,Sifakis 82]

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** [Pneuli 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, □ ("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" or "future" or "exists")

What is Model Checking?

Does model M satisfy a property P ?
(written M |= P)

What is "M"?

What is "P"?

What is "satisfy"?

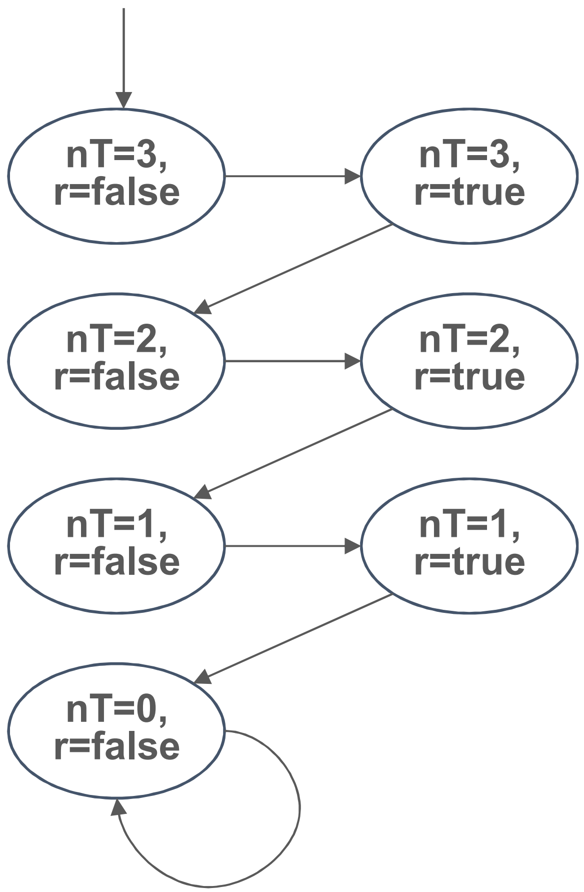
What is "M"?

Example Program:

precondition: numTickets > 0

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

State Transition Diagram



What is “M”?

Example Program:

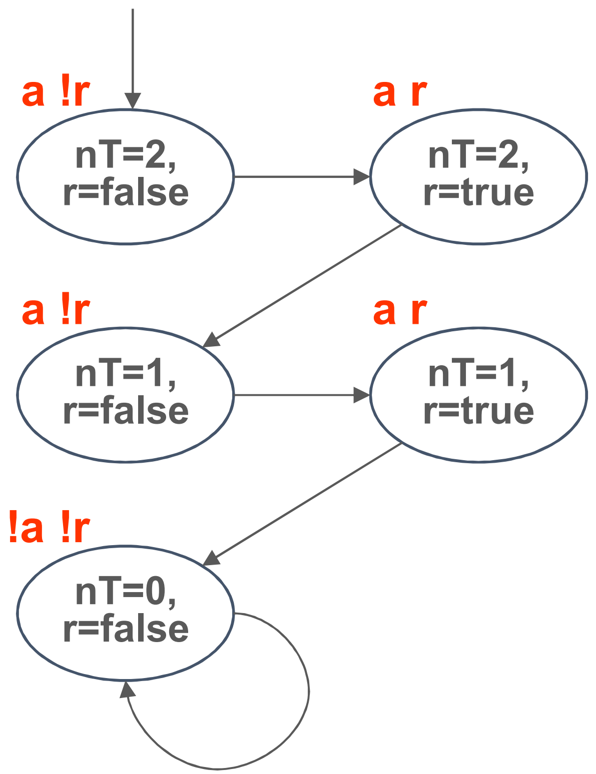
precondition: numTickets > 0

```
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

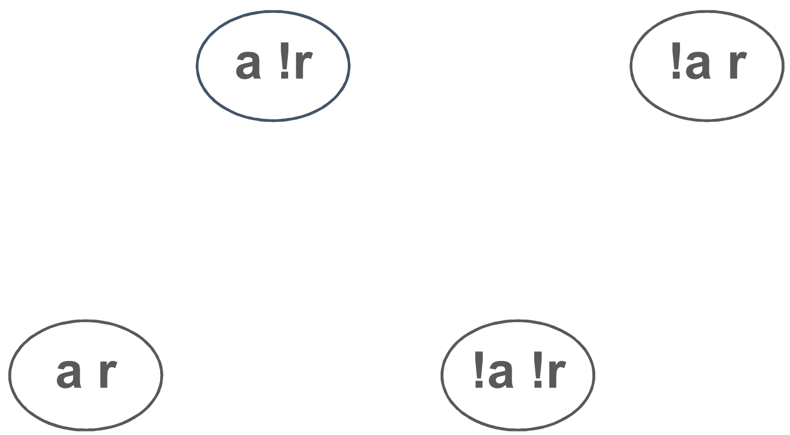
Is a ticket reserved? r



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 = ?;
    }
}
```

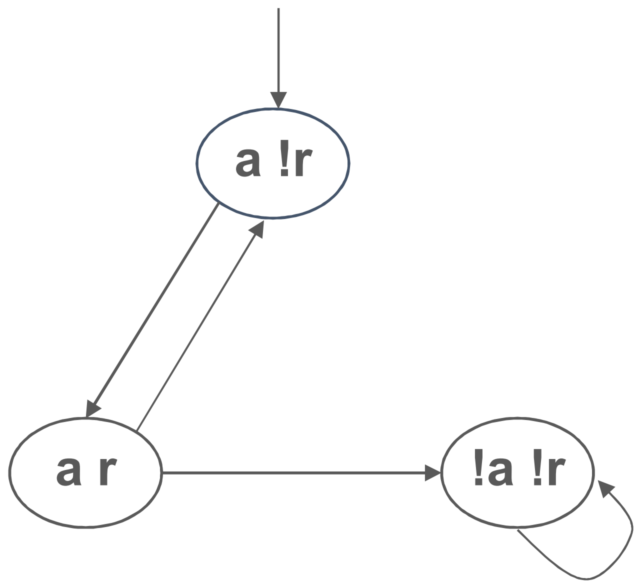


State Transition Graph or Kripke Model

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 = ?;
    }
}
```



State Transition Graph or Kripke Model

What is “M”?

State: valuations to all variables

concrete state: (numTickets=5, reserved=false)

abstract state: (a=true, r=false)

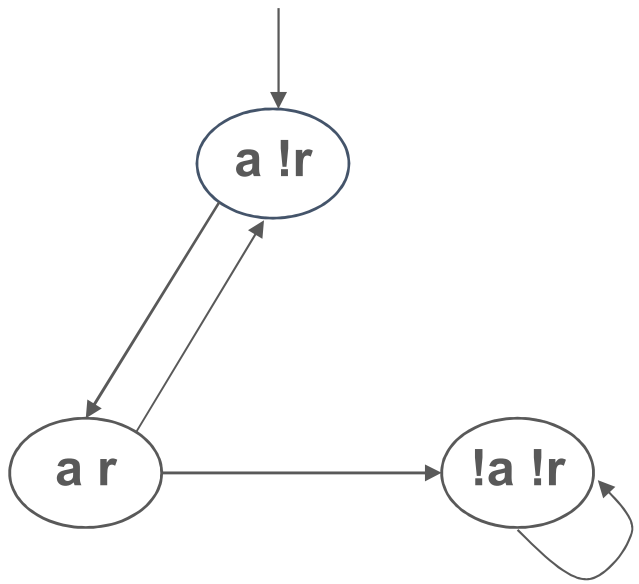
Initial states: subset of states

Arcs: transitions between states

Atomic Propositions:

a: numTickets > 0

r: reserved = true



State Transition Graph or Kripke Model

An Example Concurrent Program

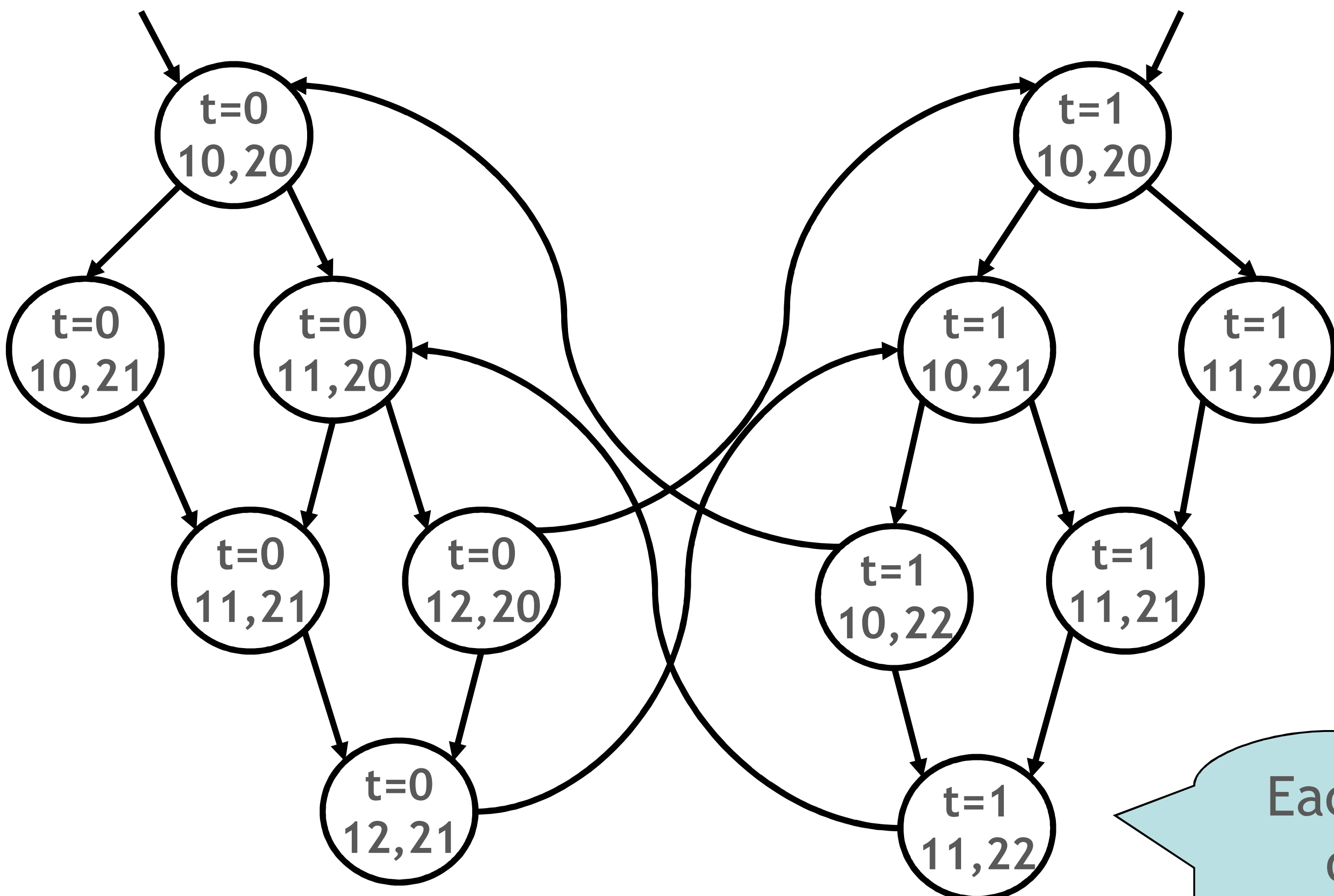
- A simple concurrent mutual exclusion program
- Two processes execute asynchronously
- There is a shared variable `turn`
- Two processes use the shared variable to ensure that they are not in the critical section at the same time
- Can be viewed as a “fundamental” program: any bigger concurrent one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

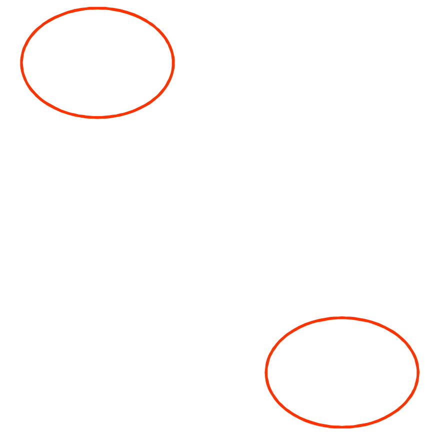


Next: formalize this intuition …

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

What is “M”? A Labelled Transition System

$$M = \langle S, S_0, R, L \rangle$$

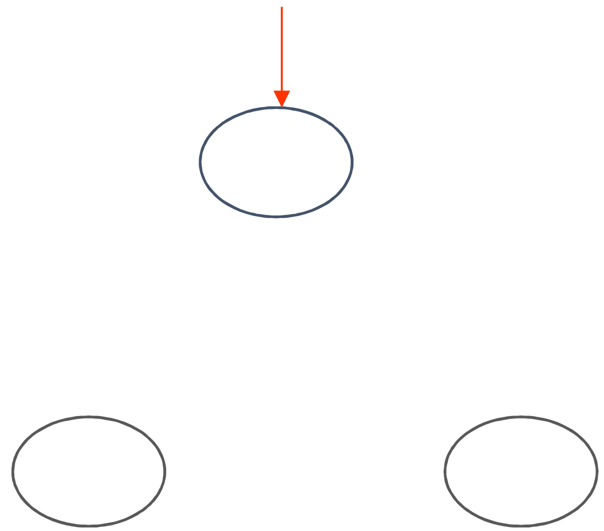


Kripke structure:

S – finite set of states

What is "M"? A Labelled Transition System

$$M = \langle S, S_0, R, L \rangle$$



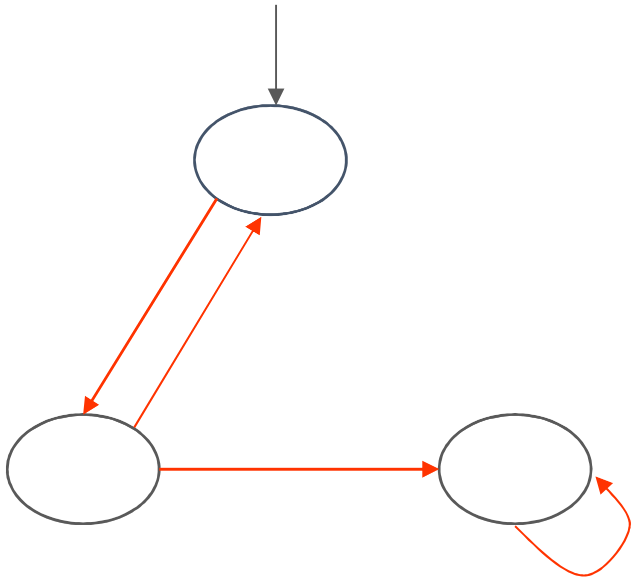
Kripke structure:

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:

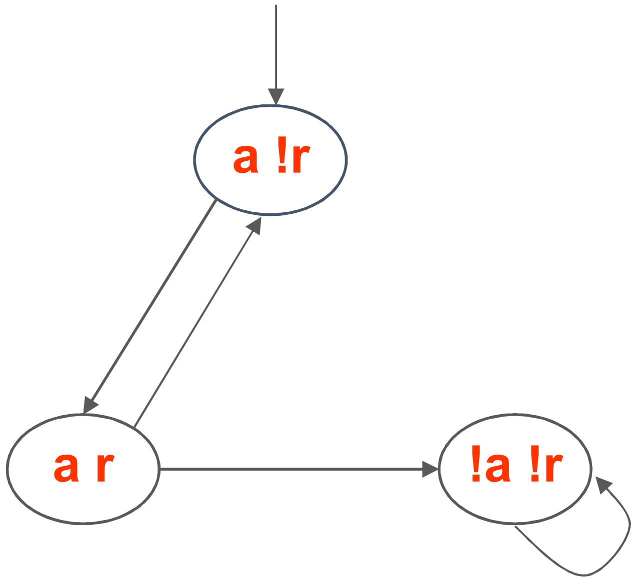
S – finite set of states

$S_0 \subseteq S$ – 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
- $L : S \rightarrow 2^{AP}$ – mapping from states to a set of atomic propositions

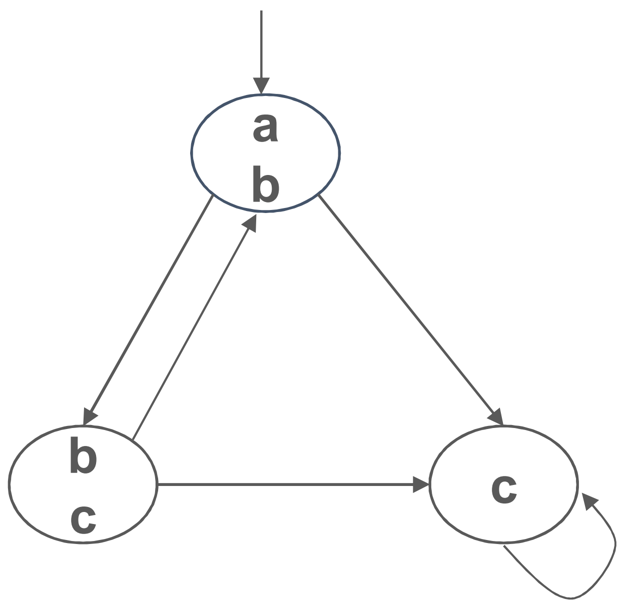
(e.g., "x=5"$\in$AP)

- 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($\neg$(pc1=12 $\wedge$ pc2=22))
- Also: “Eventually the first process enters the critical section”
  - Eventually(pc1=12)
- “pc1=12”, “pc2=22” are atomic properties

Model of Computation

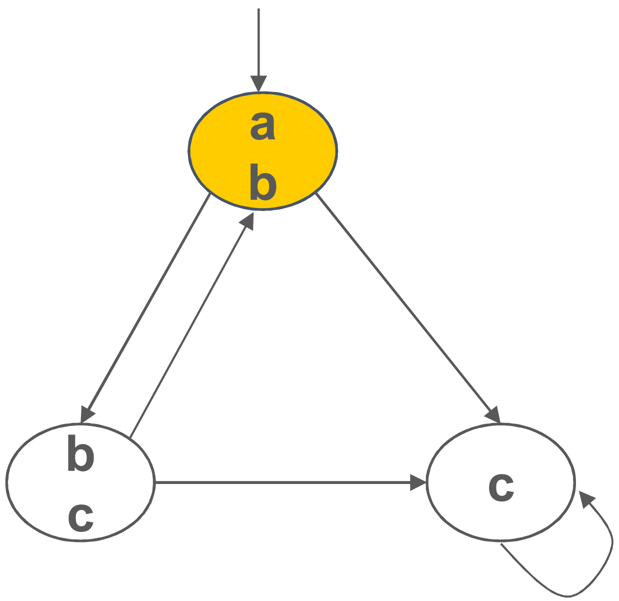


State Transition Graph

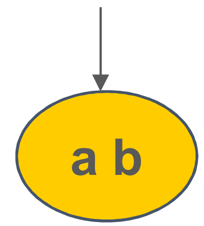
Computation Traces

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



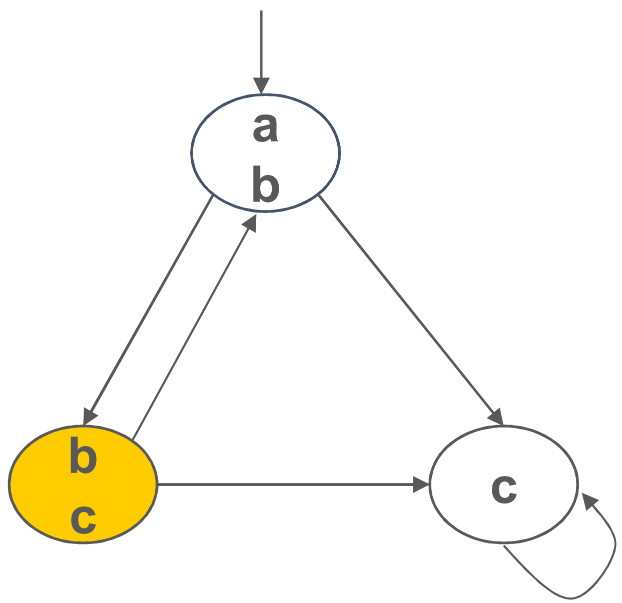
State Transition Graph



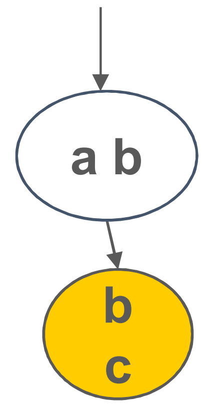
Computation Traces

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



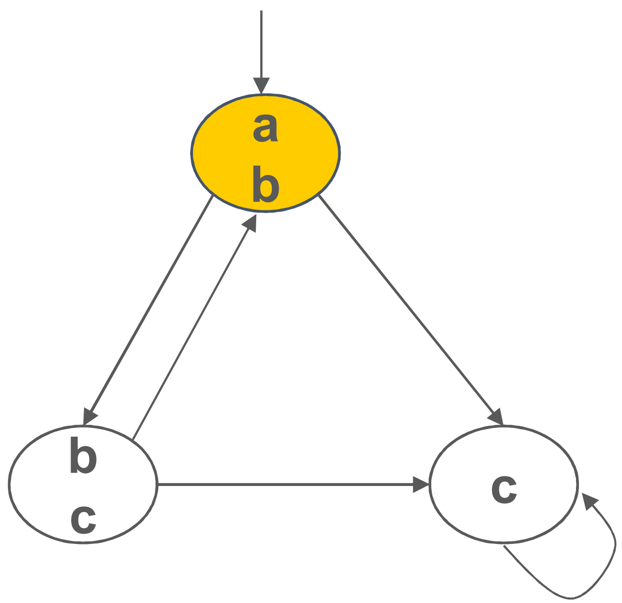
State Transition Graph



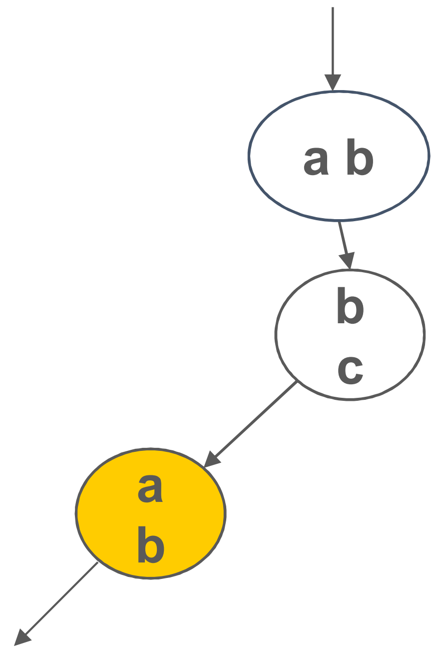
Computation Traces

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



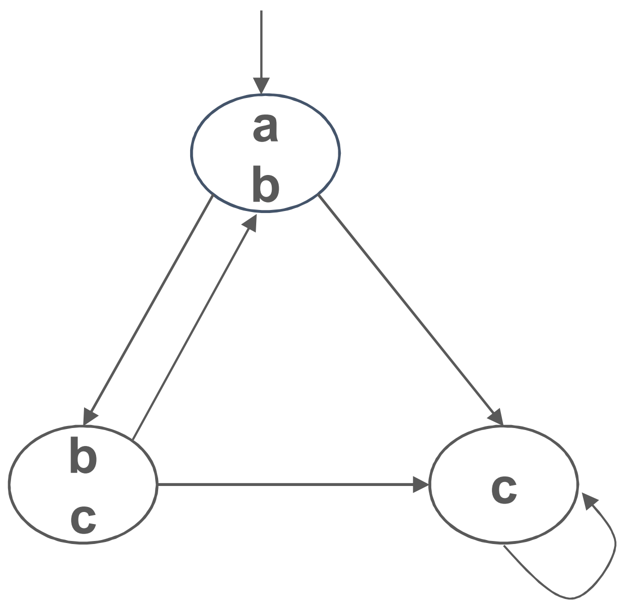
State Transition Graph



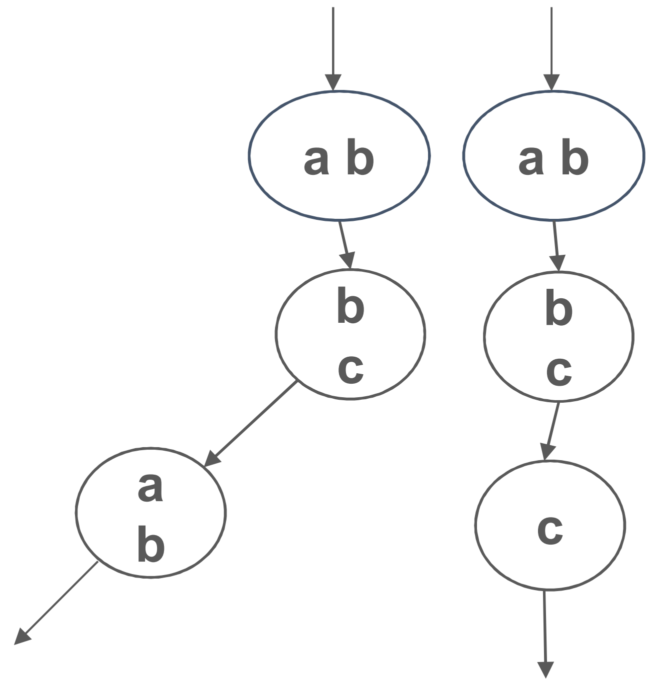
Computation Traces

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



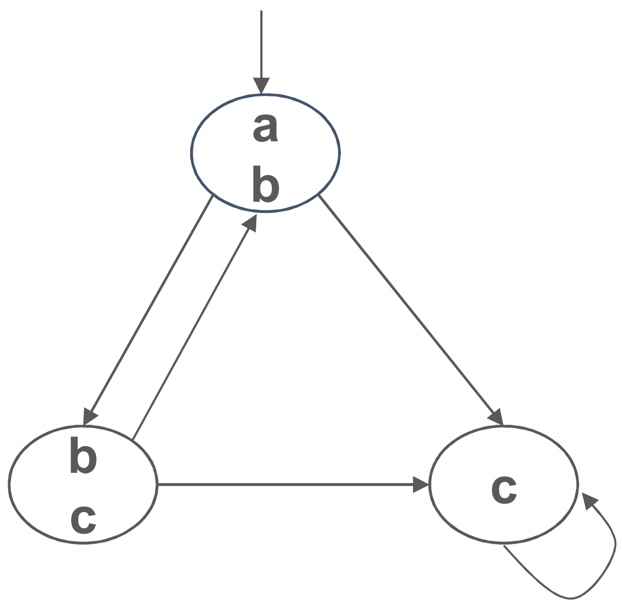
State Transition Graph



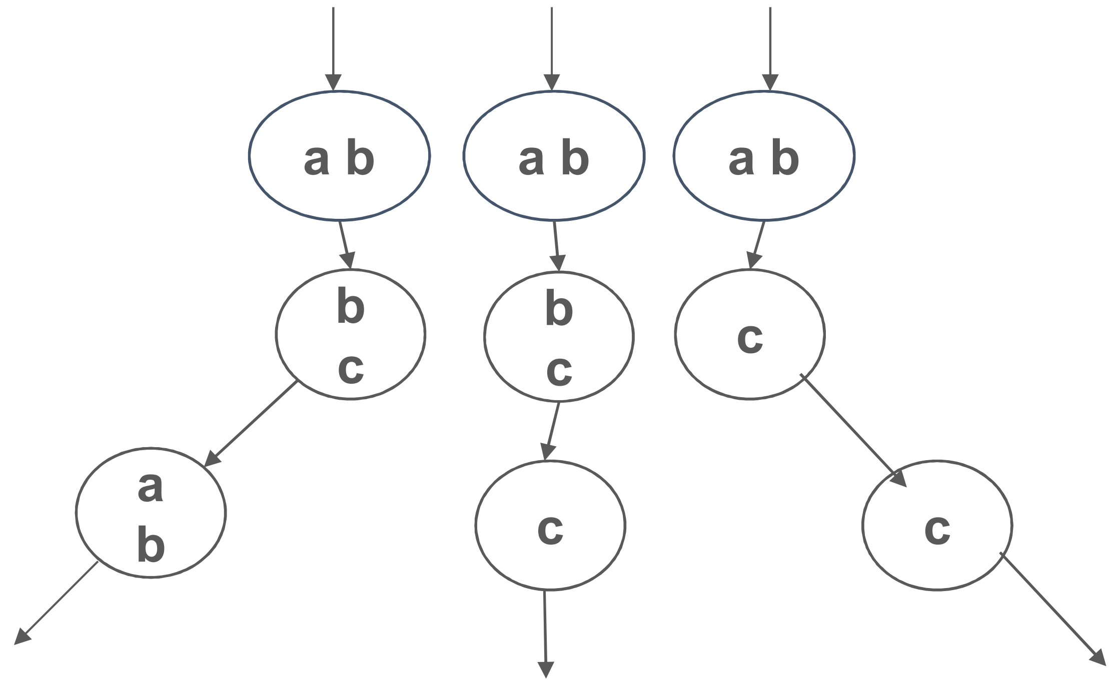
Computation Traces

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



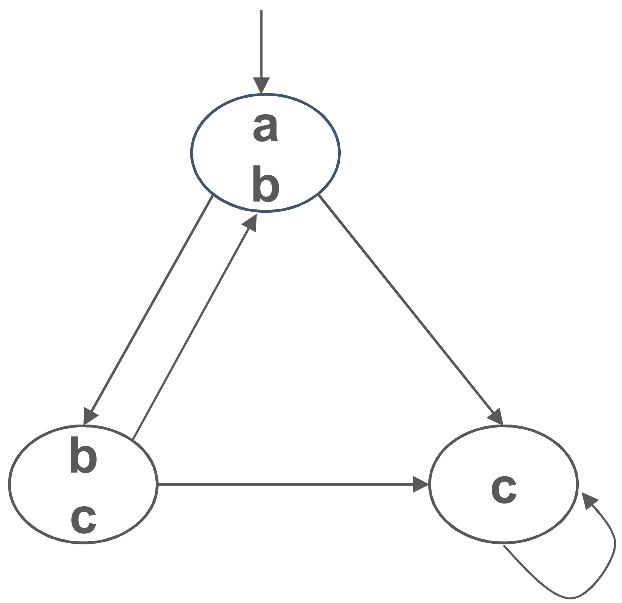
State Transition Graph



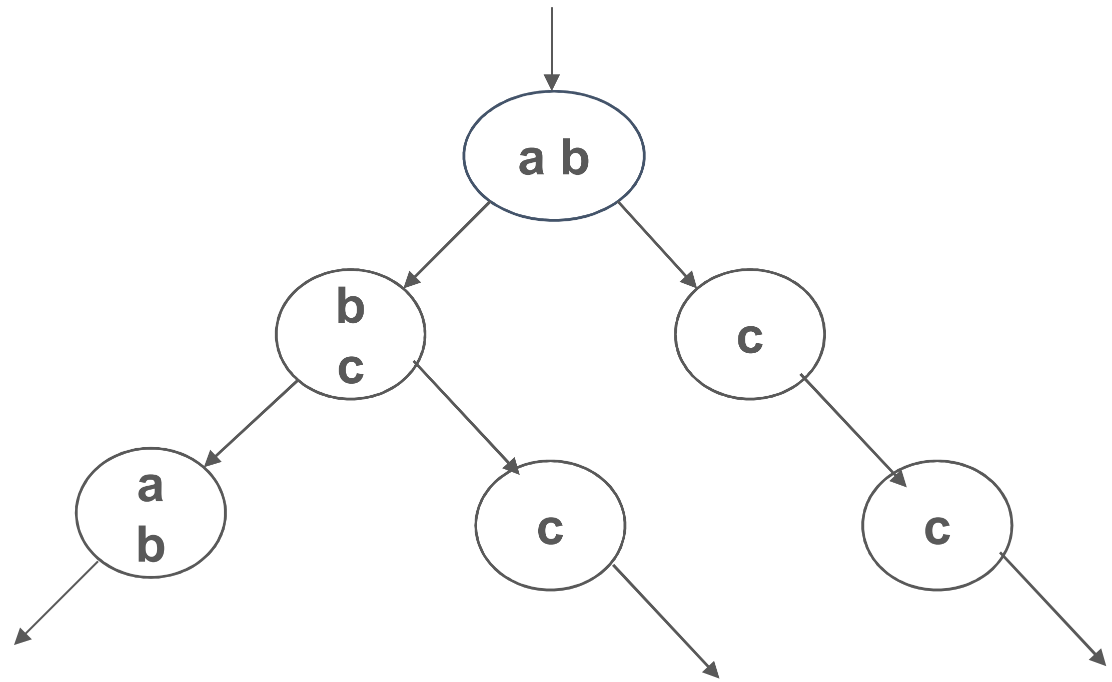
Computation Traces

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



State Transition Graph



Infinite Computation Tree

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.

Computation Tree Logics

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* = Next p, p holds in the next state
  - *G p* = Globally p, p holds in every state, p is an invariant
  - *F p* = Future p, p will hold in a future state, p holds eventually
  - *p U q* = p Until q, assertion p will hold until q holds
- Precise meaning of these temporal operators are defined on execution paths

Execution Paths

- A path π in M is an infinite sequence of states (s0, s1, s2, ...), such that $\forall i \geq 0. (s_i, s_{i+1}) \in R$
  - π^i denotes the suffix of π starting at s_i
- $M, \pi \vDash f$ means that f holds along path π in the Kripke structure M,
  - "the path π in the transition system makes the temporal logic predicate f true"
  - Example: $M, \pi \vDash G(\neg(pc1=12 \wedge pc2=22))$
- In some temporal logics one can quantify the paths starting from a state using path quantifiers
  - A : for all paths
  - E : there exists a path

Summary: Formulas over States and Paths

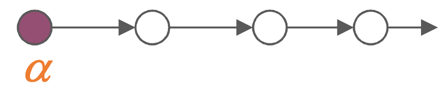
- State formulas
  - Describe a property of a state in a model M
  - If $p \in AP$, then p is a state formula
  - If f and g are state formulas, then $\neg f$, $f \wedge g$ and $f \vee g$ are state formulas
  - If f is a path formula, then **E** f and **A** f are state formulas
- Path formulas
  - Describe a property of an infinite path through a model M
  - If f is a state formula, then f is also a path formula
  - If f and g are path formulas, then $\neg f$, $f \wedge g$, $f \vee g$, **X** f, **F** f, **G** f, and f **U** g are path formulas

LTL logic operators wrt Paths

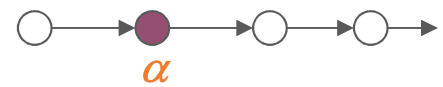
Linear Time Logic (LTL) [Pnueli 77]: logic of temporal sequences.

- LTL properties are constructed from atomic propositions in AP; logical operators $\wedge$, $\vee$, $\neg$; and temporal operators X, G, F, U.
- The semantics of LTL properties is defined on paths:

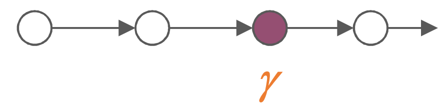
- **α: α holds in the current state (atomic)**



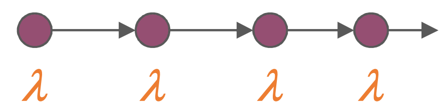
- **Xα: α holds in the next state (Next)**



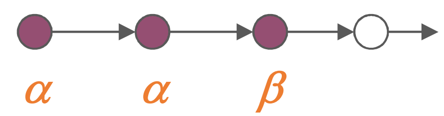
- **Fγ: γ holds eventually (Future)**



- **Gλ: λ holds from now on (Globally)**



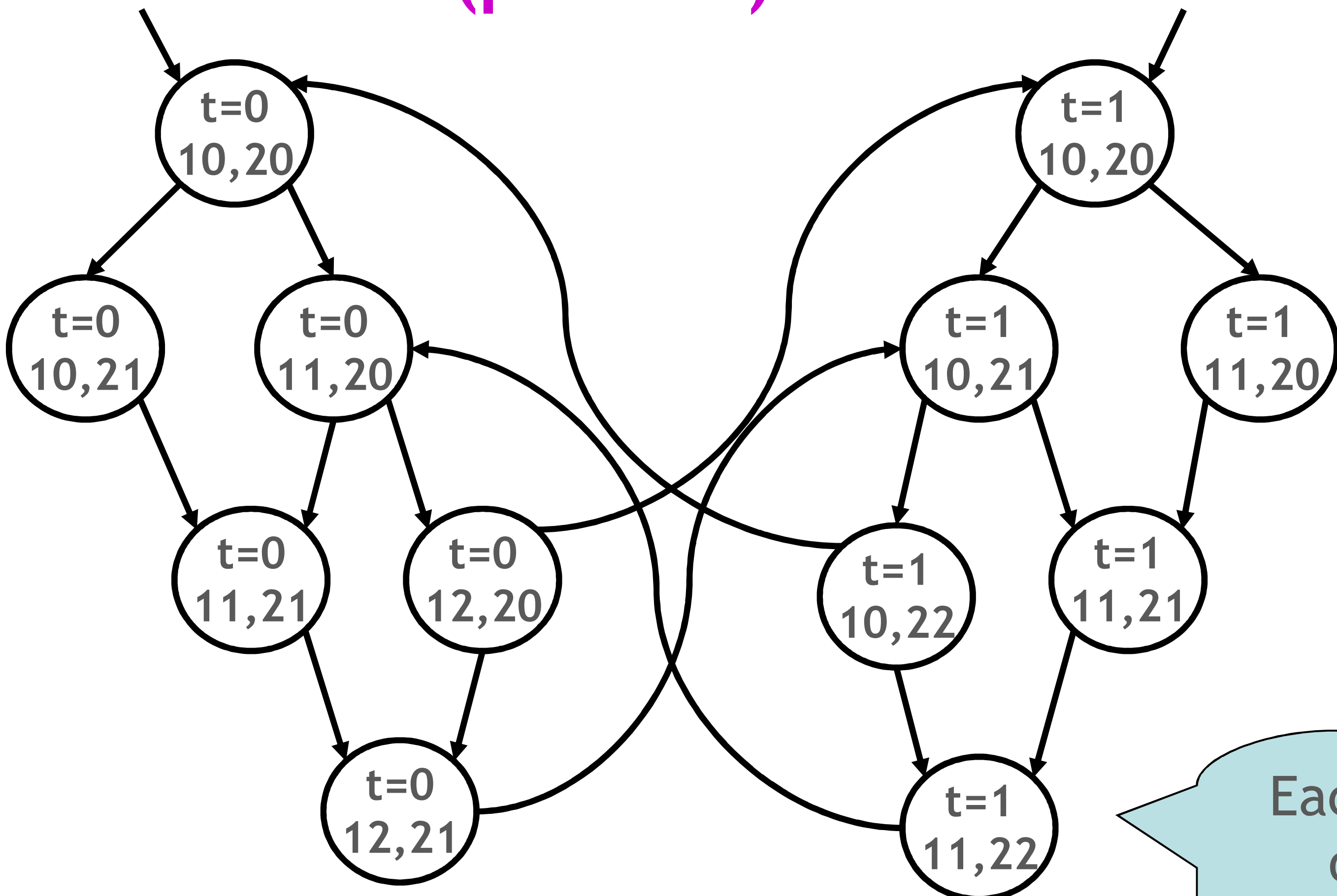
- **(α U β): α holds until β holds (Until)**



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 $\wedge$ pc2=22)):
    G($\neg$(pc1=12 $\wedge$ pc2=22))
  - *Eventually*(pc1=12):
    F(pc1=12)

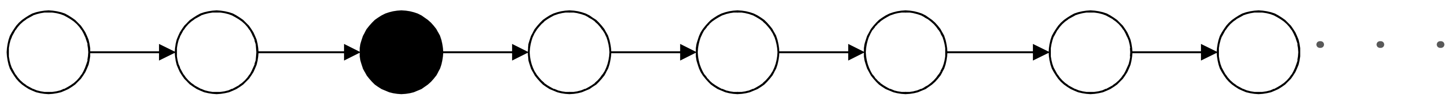
- *Invariant*(¬(pc1=12 ∧ pc2=22)):
  G(¬(pc1=12 ∧ pc2=22))
- *Eventually*(pc1=12):
  F(pc1=12)



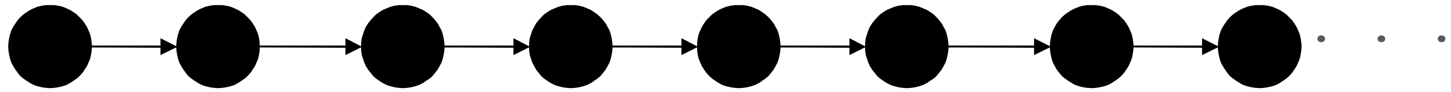
Each state is a valuation of all the variables: turn and the two program counters for two processes

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

Typical LTL Formulas

- **G** (*Req* $\Rightarrow$ **F** *Ack*): whenever *Request* occurs, it will be eventually *Acknowledged.*
- **G** (*DeviceEnabled*): *DeviceEnabled* always holds on every computation path.
- **G** (**F** *Restart*): Fairness: from any state one will eventually get to a *Restart* state. I.e. *Restart* states occur infinitely often.
- **G** (*Reset* $\Rightarrow$ **F** *Restart*): whenever the reset button is pressed one will eventually get to the *Restart* state.
- Pedantic note:
  - G is sometimes written $\square$
  - F is sometimes written $\lozenge$

Practice Writing Properties

- If the door is locked, it will not open until someone unlocks it
  - 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

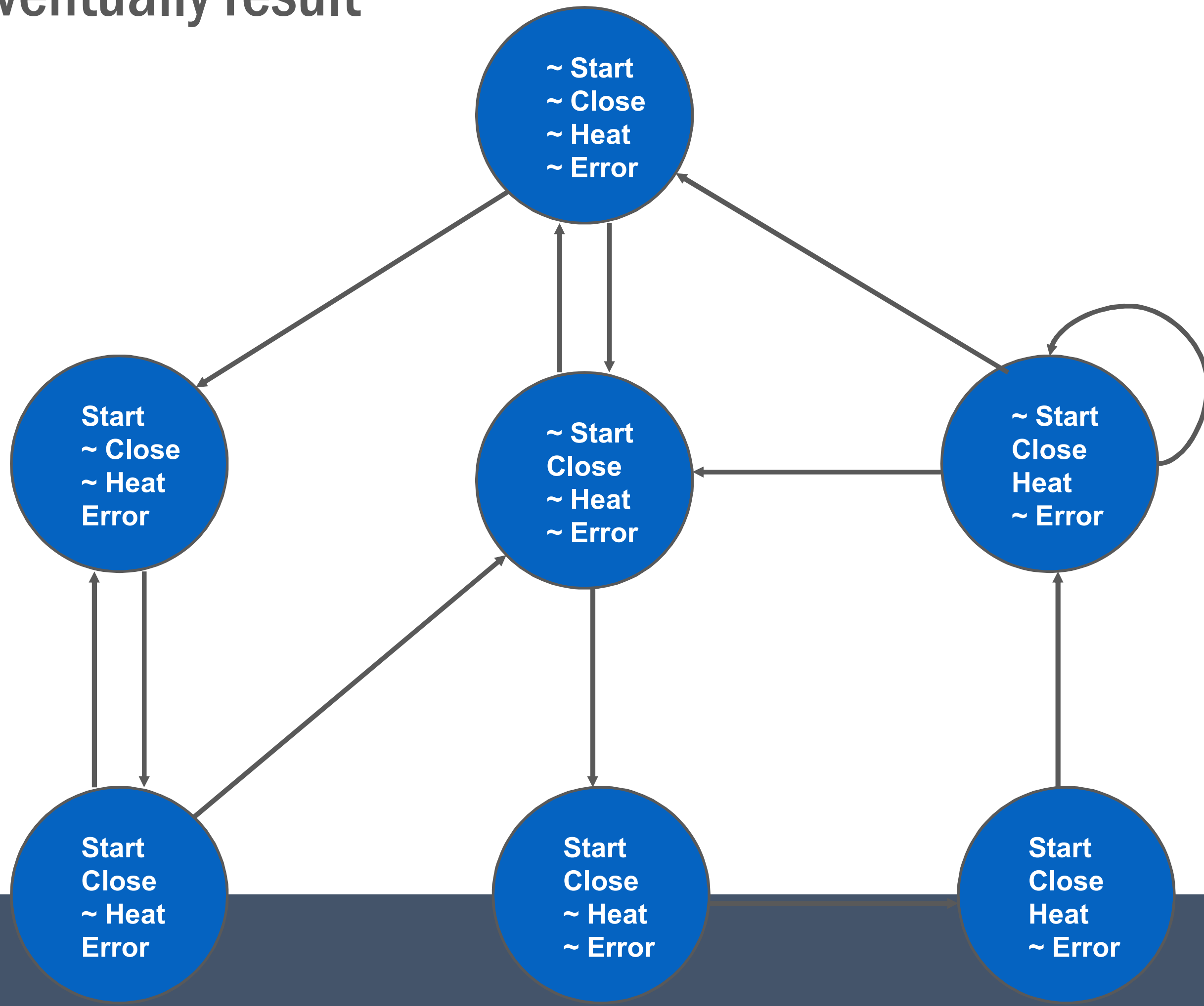
- If the door is locked, it will not open until someone unlocks it
  - assume atomic predicates locked, unlocked, open
  - G (locked $\Rightarrow$ ($\neg$open U unlocked))

- If you press ctrl-C, you will get a command line prompt
  - G (ctrlC $\Rightarrow$ F prompt)

- The saw will not run unless the safety guard is engaged
  - G ($\neg$safety $\Rightarrow$ $\neg$running)

LTL Model Checking Example

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

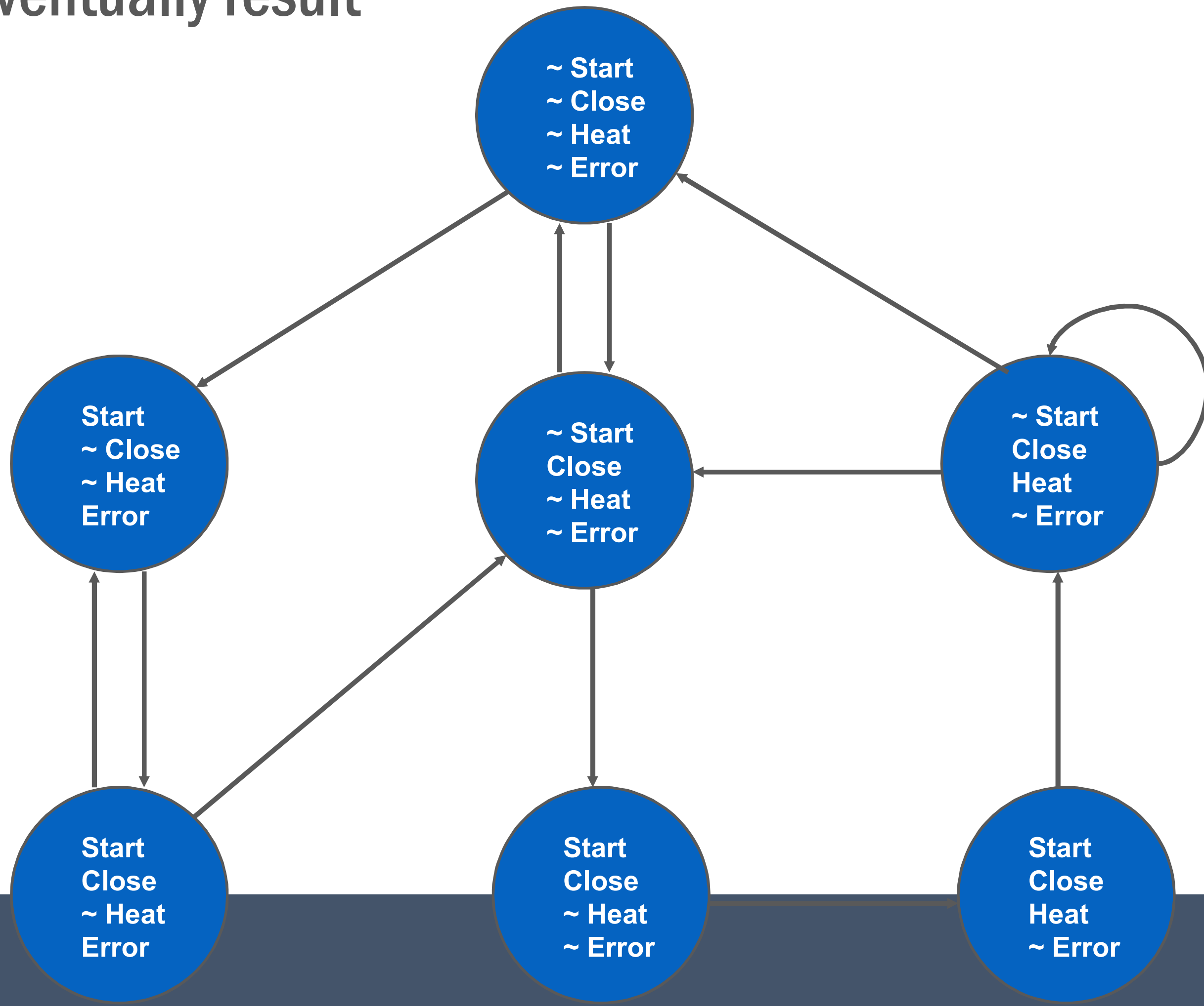


LTL Model Checking

- f (primitive formula)
  - Just check the properties of the current state
- X f
  - Verify f holds in all successors of the current state
- G f
  - Find all reachable states from the current state, and ensure f holds in all of them
    - use depth-first or breadth-first search
- f U g
  - 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.
- F f
  - 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
- G(Start $\Rightarrow$ F Heat)



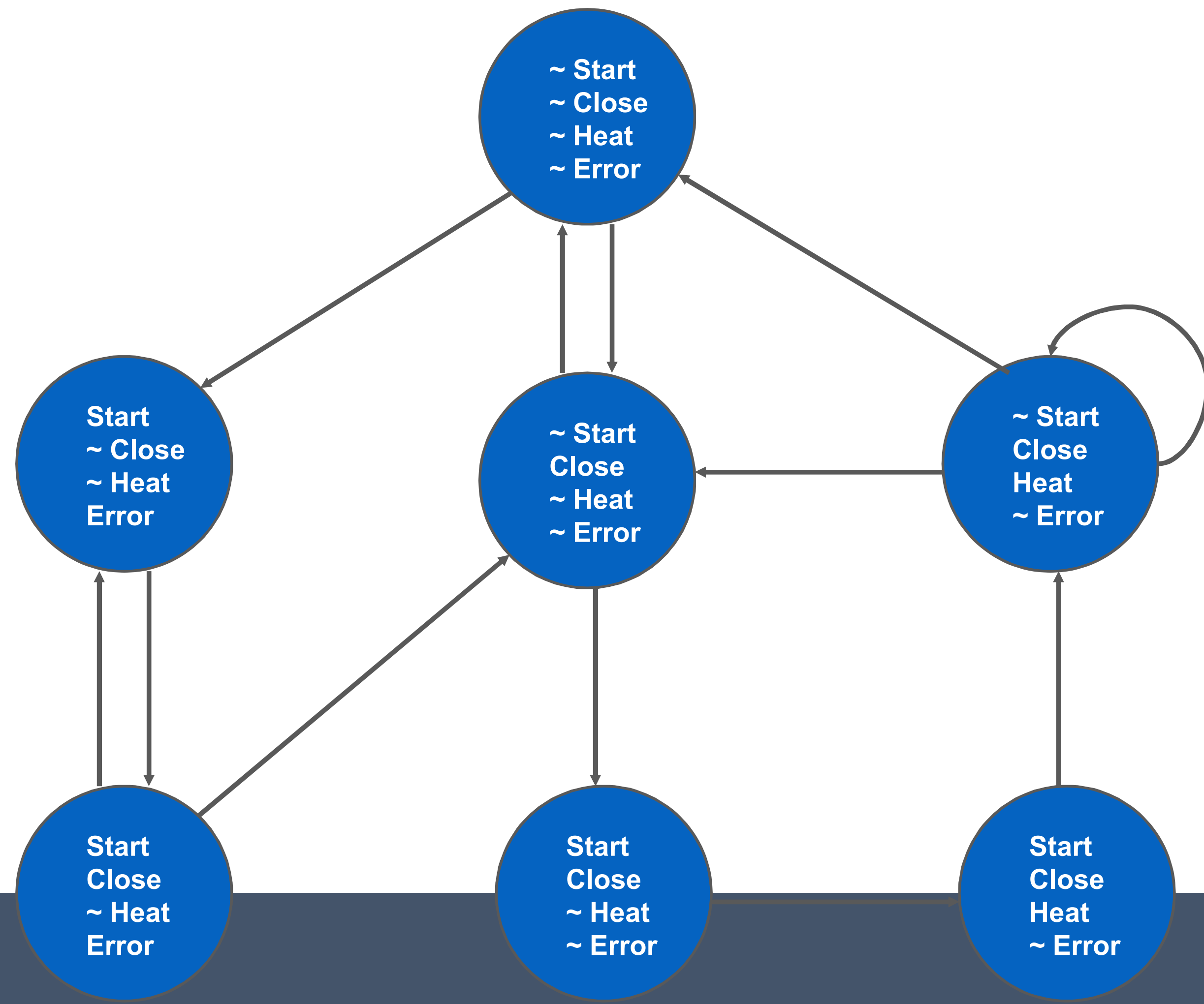
LTL Model Checking Example

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

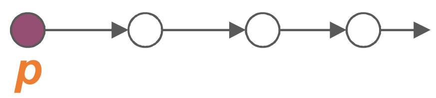
(¬Heat) U Close

(¬Heat) W Close

G (not Closed => not Heat)



Semantics of LTL Formulas



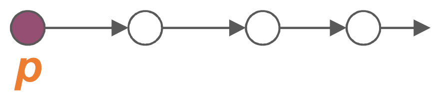
$M, \pi \vDash p \quad \Leftrightarrow \quad \pi = s\ldots \wedge p \in L(s)$

$M, \pi \vDash \neg g \quad \Leftrightarrow \quad M, \pi \nvDash g$

$M, \pi \vDash g_1 \wedge g_2 \quad \Leftrightarrow \quad M, \pi \vDash g_1 \wedge M, \pi \vDash g_2$

$M, \pi \vDash g_1 \vee g_2 \quad \Leftrightarrow \quad M, \pi \vDash g_1 \vee M, \pi \vDash g_2$

Semantics of LTL Formulas

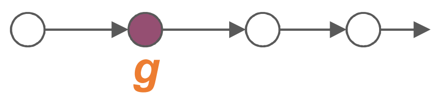


$$M, \pi \models p \quad \Leftrightarrow \quad \pi = s\ldots \wedge p \in L(s)$$

$$M, \pi \models \neg g \quad \Leftrightarrow \quad M, \pi \not\models g$$

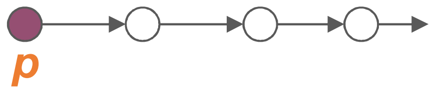
$$M, \pi \models g_1 \wedge g_2 \quad \Leftrightarrow \quad M, \pi \models g_1 \wedge M, \pi \models g_2$$

$$M, \pi \models g_1 \vee g_2 \quad \Leftrightarrow \quad M, \pi \models g_1 \vee M, \pi \models g_2$$



$$M, \pi \models \mathbf{X}\, g \quad \Leftrightarrow \quad M, \pi^1 \models g$$

Semantics of LTL Formulas

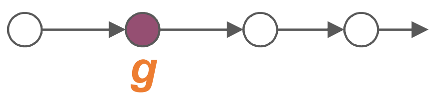


$M, \pi \vDash p \quad \Leftrightarrow \quad \pi = s\ldots \wedge p \in L(s)$

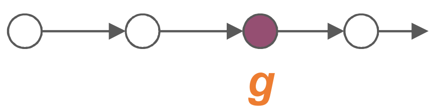
$M, \pi \vDash \neg g \quad \Leftrightarrow \quad M, \pi \nvDash g$

$M, \pi \vDash g_1 \wedge g_2 \quad \Leftrightarrow \quad M, \pi \vDash g_1 \wedge M, \pi \vDash g_2$

$M, \pi \vDash g_1 \vee g_2 \quad \Leftrightarrow \quad M, \pi \vDash g_1 \vee M, \pi \vDash g_2$

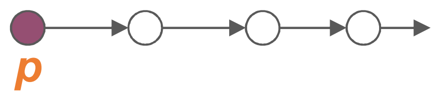


$M, \pi \vDash \mathbf{X}\, g \quad \Leftrightarrow \quad M, \pi^1 \vDash g$



$M, \pi \vDash \mathbf{F}\, g \quad \Leftrightarrow \quad \exists k \geq 0 \mid M, \pi^k \vDash g$

Semantics of LTL Formulas

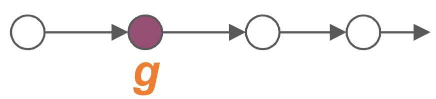


$$M, \pi \vDash p \quad \Leftrightarrow \quad \pi = s\ldots \wedge p \in L(s)$$

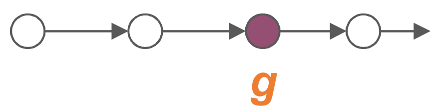
$$M, \pi \vDash \neg g \quad \Leftrightarrow \quad M, \pi \nvDash g$$

$$M, \pi \vDash g_1 \wedge g_2 \quad \Leftrightarrow \quad M, \pi \vDash g_1 \wedge M, \pi \vDash g_2$$

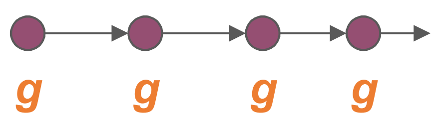
$$M, \pi \vDash g_1 \vee g_2 \quad \Leftrightarrow \quad M, \pi \vDash g_1 \vee M, \pi \vDash g_2$$



$$M, \pi \vDash \mathbf{X}\, g \quad \Leftrightarrow \quad M, \pi^1 \vDash g$$

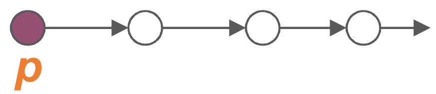


$$M, \pi \vDash \mathbf{F}\, g \quad \Leftrightarrow \quad \exists k \geq 0 \mid M, \pi^k \vDash g$$



$$M, \pi \vDash \mathbf{G}\, g \quad \Leftrightarrow \quad \forall k \geq 0 \mid M, \pi^k \vDash g$$

Semantics of LTL Formulas

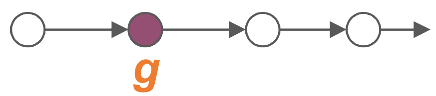


$M, \pi \vDash p \quad \Leftrightarrow \quad \pi = s\ldots \wedge p \in L(s)$

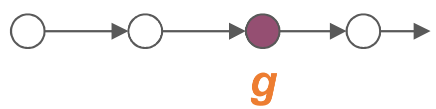
$M, \pi \vDash \neg g \quad \Leftrightarrow \quad M, \pi \nvDash g$

$M, \pi \vDash g_1 \wedge g_2 \quad \Leftrightarrow \quad M, \pi \vDash g_1 \wedge M, \pi \vDash g_2$

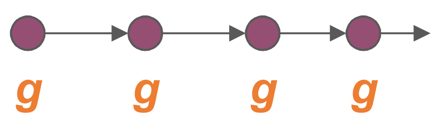
$M, \pi \vDash g_1 \vee g_2 \quad \Leftrightarrow \quad M, \pi \vDash g_1 \vee M, \pi \vDash g_2$



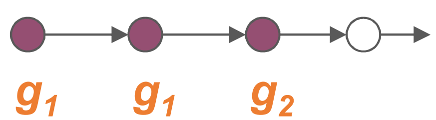
$M, \pi \vDash \mathbf{X}\, g \quad \Leftrightarrow \quad M, \pi^1 \vDash g$



$M, \pi \vDash \mathbf{F}\, g \quad \Leftrightarrow \quad \exists k \geq 0 \mid M, \pi^k \vDash g$

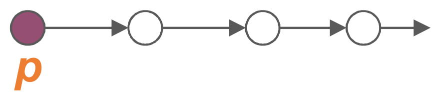


$M, \pi \vDash \mathbf{G}\, g \quad \Leftrightarrow \quad \forall k \geq 0 \mid M, \pi^k \vDash g$



$M, \pi \vDash g_1 \mathbf{U}\, g_2 \quad \Leftrightarrow \quad \exists k \geq 0 \mid M, \pi^k \vDash g_2 \wedge \forall 0 \leq j < k \; M, \pi^j \vDash g_1$

Semantics of LTL Formulas

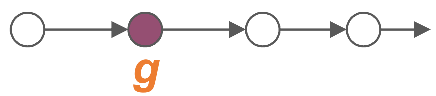


$M, \pi \vDash p \quad \Leftrightarrow \quad \pi = s\ldots \wedge p \in L(s)$

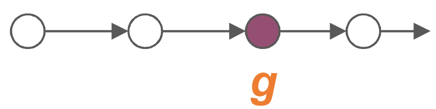
$M, \pi \vDash \neg g \quad \Leftrightarrow \quad M, \pi \nvDash g$

$M, \pi \vDash g_1 \wedge g_2 \quad \Leftrightarrow \quad M, \pi \vDash g_1 \wedge M, \pi \vDash g_2$

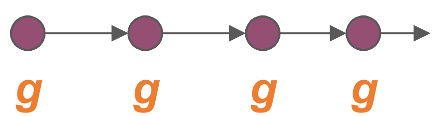
$M, \pi \vDash g_1 \vee g_2 \quad \Leftrightarrow \quad M, \pi \vDash g_1 \vee M, \pi \vDash g_2$



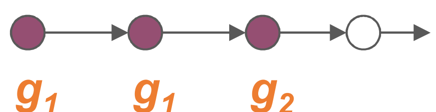
$M, \pi \vDash \mathbf{X}\, g \quad \Leftrightarrow \quad M, \pi^1 \vDash g$



$M, \pi \vDash \mathbf{F}\, g \quad \Leftrightarrow \quad \exists k \geq 0 \mid M, \pi^k \vDash g$



$M, \pi \vDash \mathbf{G}\, g \quad \Leftrightarrow \quad \forall k \geq 0 \mid M, \pi^k \vDash g$



$M, \pi \vDash g_1 \mathbf{U}\, g_2 \quad \Leftrightarrow \quad \exists k \geq 0 \mid M, \pi^k \vDash g_2 \wedge \forall 0 \leq j < k\ M, \pi^j \vDash g_1$

g_2 must eventually hold

semantics of "until" in English are potentially unclear—
that's why we have a formal definition

Semantics of Formulas

$M, s \models p$	$\Leftrightarrow p \in L(s)$	$M, \pi \models f$	$\Leftrightarrow \pi = s\ldots \wedge M, s \models f$
$M, s \models \neg f$	$\Leftrightarrow M, s \not\models f$	$M, \pi \models \neg g$	$\Leftrightarrow M, \pi \not\models g$
$M, s \models f_1 \wedge f_2$	$\Leftrightarrow M, s \models f_1 \wedge M, s \models f_2$	$M, \pi \models g_1 \wedge g_2$	$\Leftrightarrow M, \pi \models g_1 \wedge M, \pi \models g_2$
$M, s \models f_1 \vee f_2$	$\Leftrightarrow M, s \models f_1 \vee M, s \models f_2$	$M, \pi \models g_1 \vee g_2$	$\Leftrightarrow M, \pi \models g_1 \vee M, \pi \models g_2$
$M, s \models \mathbf{E}\, g_1$	$\Leftrightarrow \exists \pi = s\ldots \mid M, \pi \models g_1$	$M, \pi \models \mathbf{X}\, g$	$\Leftrightarrow M, \pi^1 \models g$
$M, s \models \mathbf{A}\, g_1$	$\Leftrightarrow \forall \pi = s\ldots\ M, \pi \models g_1$	$M, \pi \models \mathbf{F}\, g$	$\Leftrightarrow \exists k \geq 0 \mid M, \pi^k \models g$
		$M, \pi \models \mathbf{G}\, g$	$\Leftrightarrow \forall k \geq 0 \mid M, \pi^k \models g$
		$M, \pi \models g_1 \mathbf{U}\, g_2$	$\Leftrightarrow \exists k \geq 0 \mid M, \pi^k \models g_2$
			$\wedge \forall 0 \leq j < k\ M, \pi^j \models g_1$

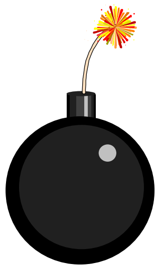
Model Checking Complexity

- 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*)



$$M = M_1 \mid\mid \ldots \mid\mid M_n$$

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.

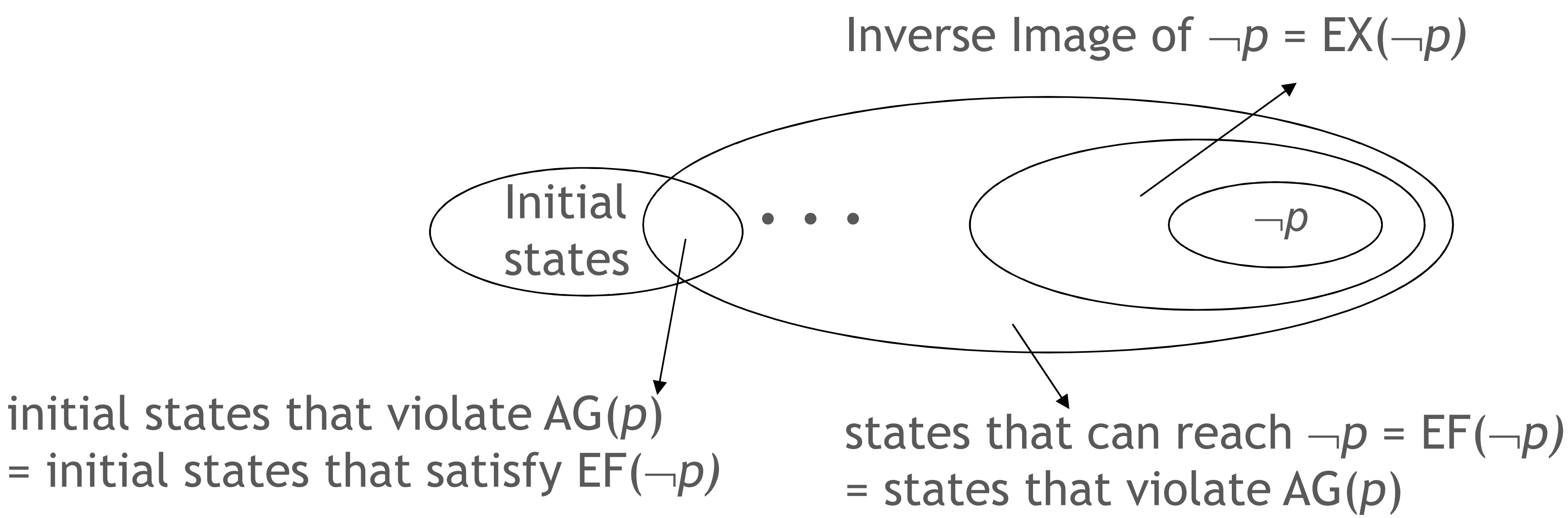
Temporal Properties ≡ Fixpoints

- 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 **fixpoint** of Func: $2^S \rightarrow 2^S$
- Given $Z \subseteq S$,
  - Func(Z) = ¬p ∪ **reach-in-one-step**(Z)
  - or Func(Z) = ¬p ∪ EX(Z)

This is called the ***inverse image*** *of Z*

- Actually, EF(¬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=∅, and apply the Func until you reach a fixpoint
  - This can be computed (unlike most other fixpoints)

Pictorial Backward Fixpoint



This fixpoint computation can be used for:

- verification of EF($\neg$p)
- or falsification of AG(p)

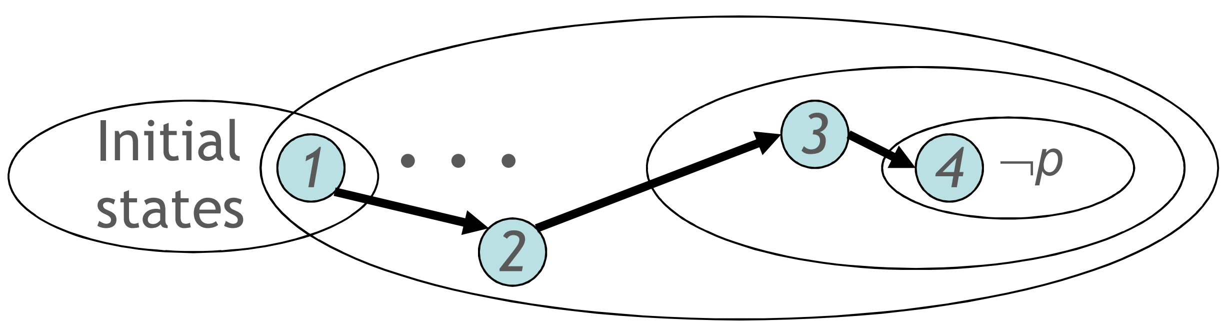
… and a similar forward fixpoint handles the other cases

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 $Z \subseteq S$
- 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., Ring#3 = 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:
  - Model Checkers today can routinely handle systems with between 100 and 300 state variables.
  - Systems with 10^{120} reachable states have been checked.
  - By using appropriate abstraction techniques, systems with an essentially **unlimited number of states** can be checked.
- Notable examples:
  - **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
  - **IEEE Futurebus** – In 1992 Clarke's group at CMU found previously undetected design errors
  - **PowerScale multiprocessor** (processor, memory controller, and bus arbiter) was verified by Verimag researchers using CAESAR toolbox
  - **Lucent telecom.** protocols were verified by FormalCheck – errors leading to lost transitions were identified
  - **PowerPC 620 Microprocessor** was verified by Motorola's Verdict model checker.

Efficient Algorithms for LTL Model Checking

- Use Büchi automata
  - Beyond the scope of this course

- Canonical reference on Model Checking:
  - Edmund Clarke, Orna Grumberg, and Doron A. Peled. Model Checking. MIT Press, 1999.

Computation Tree Logics

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

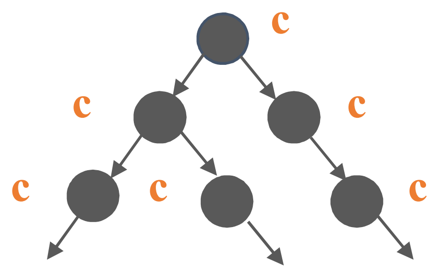
1. ***Path Quantifiers:***
   - **A** – "for every path"
   - **E** – "there exists a path"

LTL: start with an A and then use only Temporal Operators

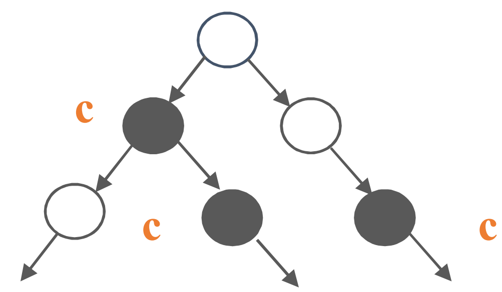
2. ***Temporal Operator:***
   - **X**α - α holds **next** time
   - **F**α - α holds sometime in the **future**
   - **G**α - α holds **globally** in the **future**
   - α **U**β - α holds **until** β holds

The Logic CTL

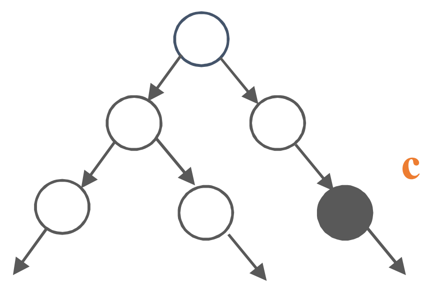
In a branching-time logic (CTL), the temporal operators quantify over the paths that are possible from a given state (s_0). Requires each temporal operator (**X**, **F**, **G**, and **U**) to be preceded by a path quantifier (**A** or **E**).



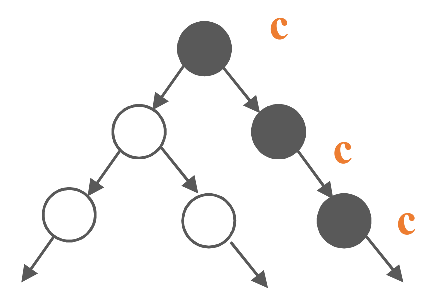
$M, s_0 \models \mathbf{AG}\ c$



$M, s_0 \models \mathbf{AF}\ c$

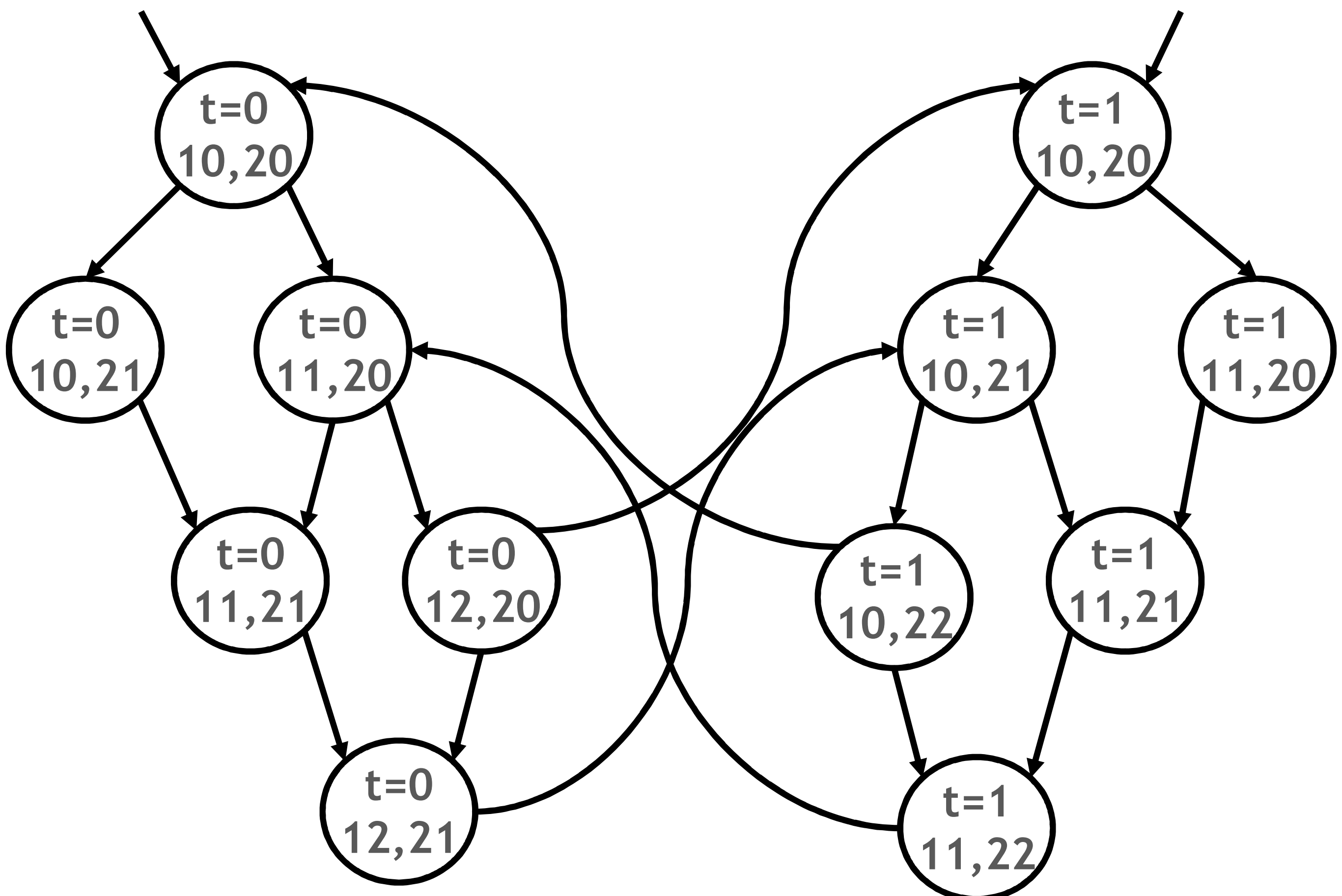


$M, s_0 \models \mathbf{EF}\ c$



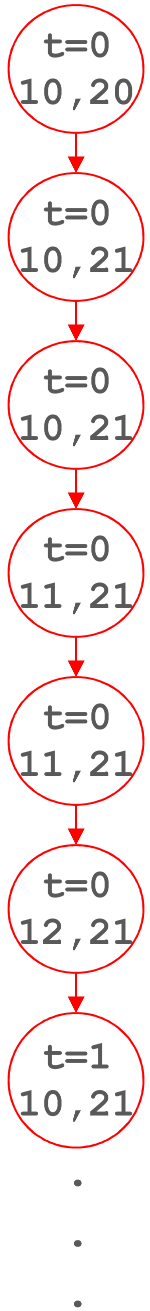
$M, s_0 \models \mathbf{EG}\ c$

Remember the Example



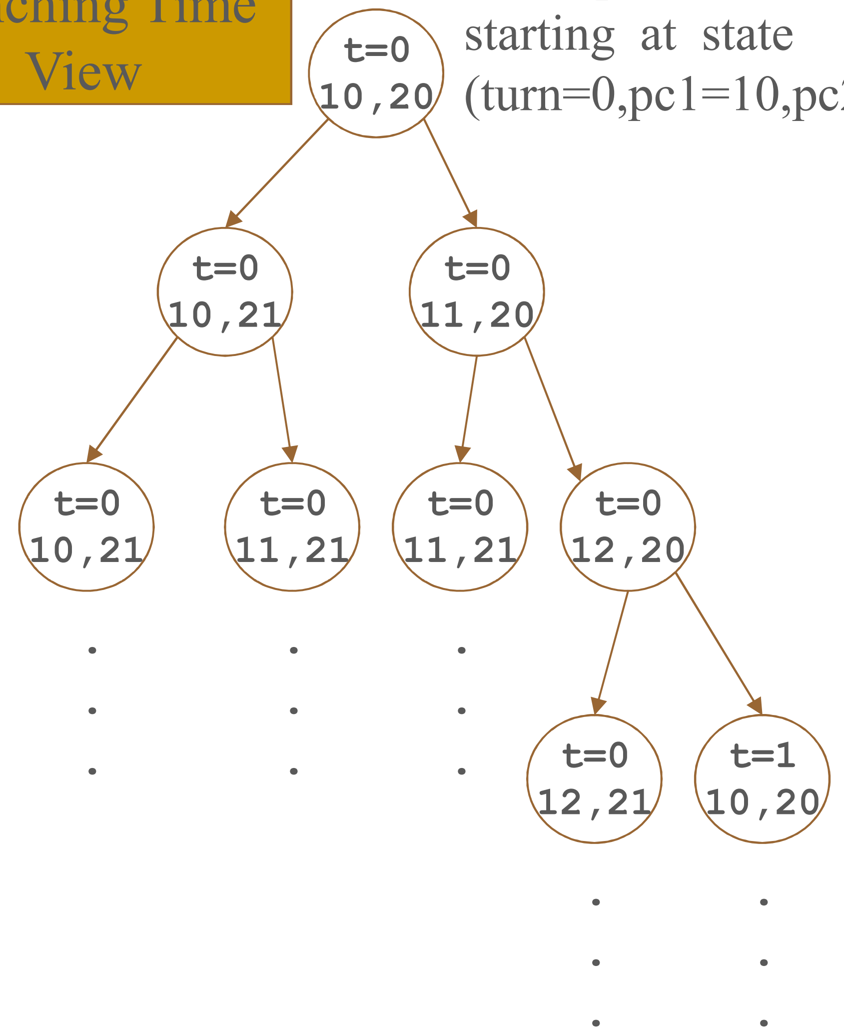
Linear vs. Branching Time

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



Linear Time View

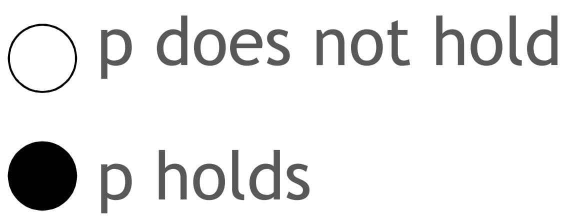
Branching Time View



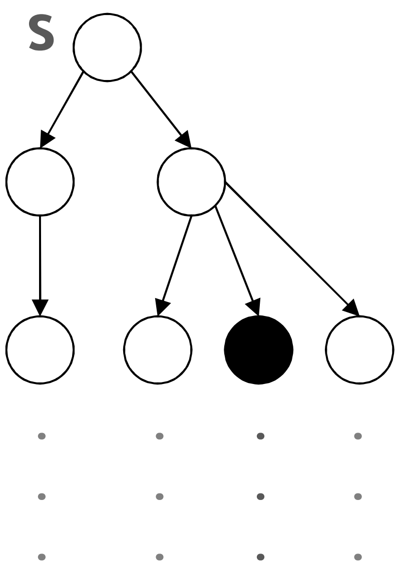
A computation tree starting at state (turn=0,pc1=10,pc2=20)

Example/Typical CTL Formulas

- **EF** (*Started* $\wedge$ $\neg$*Ready*): it is possible to get to a state where *Started* holds but *Ready* does not hold.
- **AG** (*Req* $\Rightarrow$ **AF** *Ack*): whenever *Request* occurs, it will be eventually *Acknowledged.*
- **AG** (*DeviceEnabled*): *DeviceEnabled* always holds on every computation path.
- **AG** (**EF** *Restart*): from any state it is possible to get to the *Restart* state.



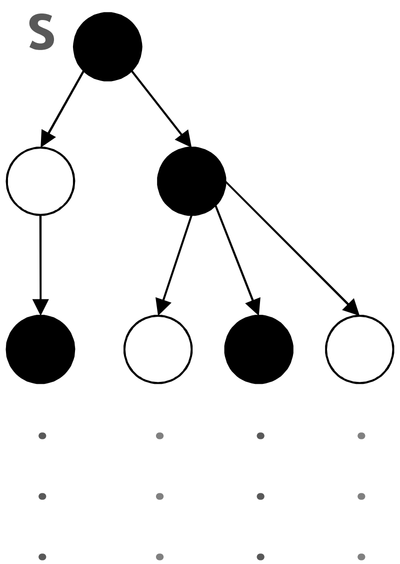
CTL Examples



At state s:

EF p, EX (EX p),
AF (¬p), ¬p holds

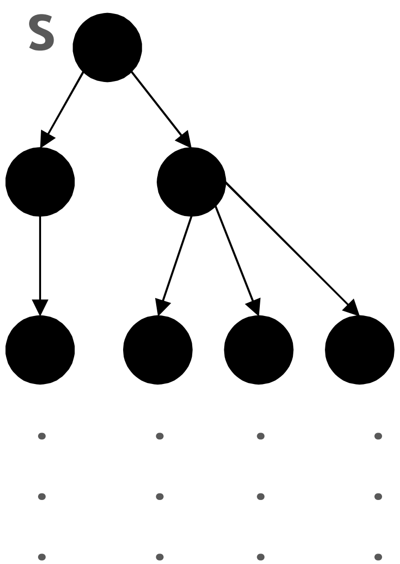
AF p, AG p,
AG (¬p), EX p,
EG p, p does not hold



At state s:

EF p, AF p,
EX (EX p),
EX p, EG p, p holds

AG p, AG (¬p),
AF (¬p) does not hold



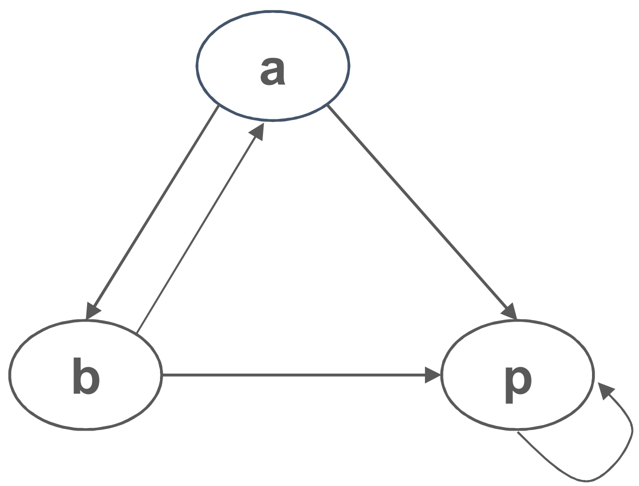
At state s:

EF p, AF p,
AG p, EG p,
Ex p, AX p, p holds

EG (¬ p), EF (¬p),
does not hold

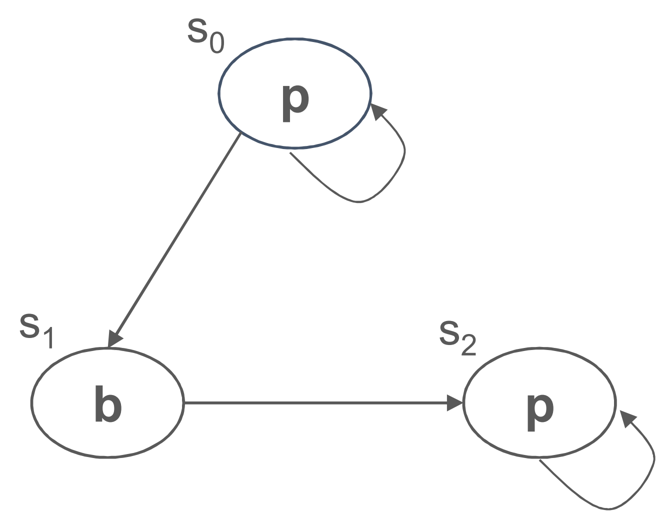
Trivia

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



Trivia

- A(FG p) cannot be expressed in CTL
  - 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
  - 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 s_0



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)

Linear vs Branching-Time 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
  - $M \models \psi \Rightarrow M' \models \psi$
- "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 (anecdotally) 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 μ-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