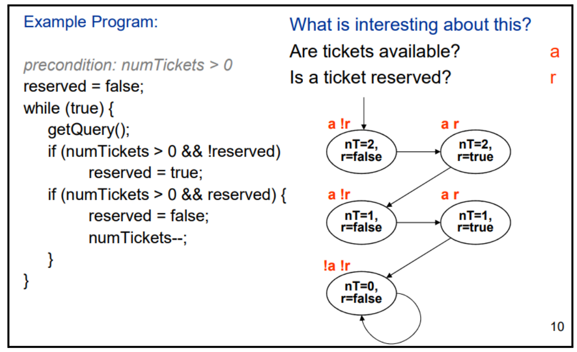
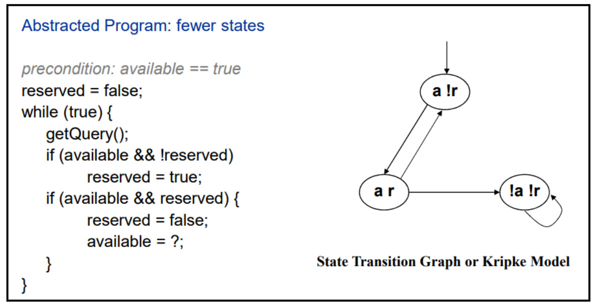
Model Checking
An automated technique used for verifying finite-state systems against specified properties.
It systematically explores all possible states of a system to verify if certain properties hold true.
✔ State Space:
The set of all possible states that a system can be in.
In a program, the state can be described by the set of all variable addresses and values.
Modifying a variable's value results in a new state.
Model checking over a program may require an additional layer of abstraction (e.g., predicate abstraction) to reduce the number of states to a manageable size.
✔ Transitions:
These are rules that describe how the system moves between states.
Each state can transition to another based on specific conditions or actions.
✔ Properties:
These are specifications that define the desired behavior of the system.
Model checking verifies whether the system satisfies these properties.
✔ Temporal Logic:
Temporal logic is the language used to express the properties to be verified.
Examples include Linear Temporal Logic (LTL) and Computation Tree Logic (CTL), which help express the system’s behavior over time.

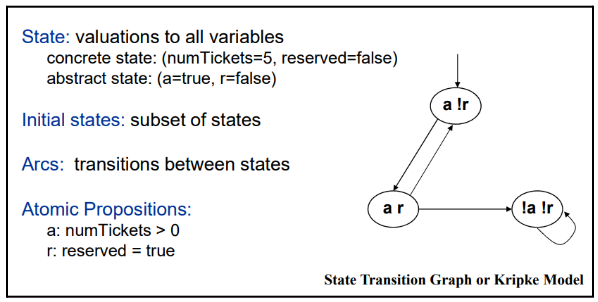
✔ Simple Example: Traffic Light Controller
Imagine a traffic light controller with three states:
States: {RED, YELLOW, GREEN}
Initial State: RED
Transitions:RED → GREEN, GREEN → YELLOW, YELLOW → RED
In this system, the traffic light switches from red to green, from green to yellow, and from yellow back to red, repeatedly.

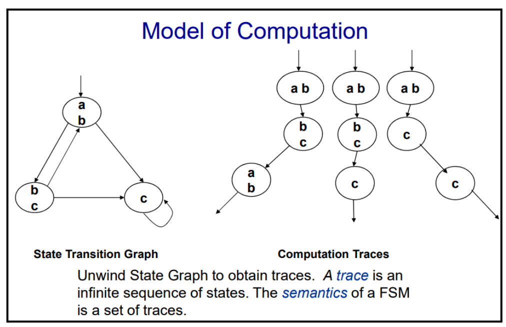
Model checking would explore all possible states of this traffic light controller to verify whether it behaves according to its specifications (e.g., ensuring that the light always transitions in this exact sequence, without errors or violations of the system's rules).

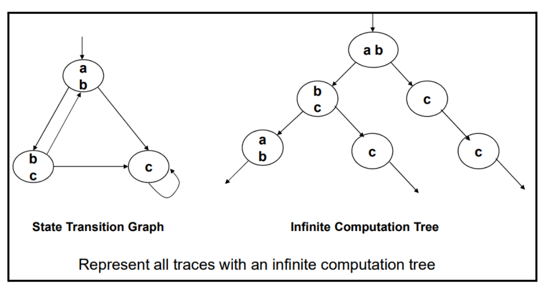
📌 Model checking is a powerful technique for verifying system properties by exhaustively exploring all states and transitions within the system. It uses temporal logic to express and check properties, ensuring that the system behaves as expected under all possible conditions.

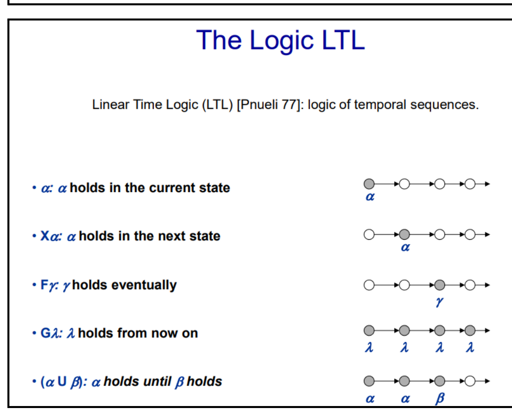












Typical LTL Formulas

- **G** (*Req* ⇒ **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* ⇒ **F** *Restart*): whenever the reset button is pressed one will eventually get to the *Restart* state.

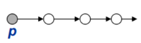
LTL Conventions

- G is sometimes written □
- F is sometimes written ◇

Notation

- A path π in M is an infinite sequence of states $s_0, s_1, \ldots$ such that for every $i \geq 0$, $(s_i, s_{i+1}) \in R$
- 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

Semantics of LTL Formulas

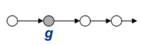


$M, \pi \vDash p \quad \Leftrightarrow \quad \pi = s \ldots \wedge p \in L(s)$

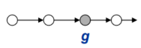
$M, \pi \vDash \neg g \quad \Leftrightarrow \quad M, \pi \nvDash g$

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

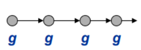
$M, \pi \vDash g_1 \vee g_2 \quad \Leftrightarrow \quad M, \pi \vDash g_1 \vee M, \pi \vDash g_2$



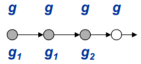
$M, \pi \vDash \mathbf{X}\, g \quad \Leftrightarrow \quad M, \pi^1 \vDash g$



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



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



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

g2 must eventually hold

$\wedge\ \forall 0 \leq j < k\ M, \pi^j \vDash g_1$

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

Practice Writing Properties

- If the door is locked, it will not open until someone unlocks it
  - assume atomic predicates locked, unlocked, open
  - G (locked ⇒ (¬open U unlocked))
- If you press ctrl-C, you will get a command line prompt
  - G (ctrlC ⇒ F prompt)
- The saw will not run unless the safety guard is engaged
  - G (¬safety ⇒ ¬running)

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

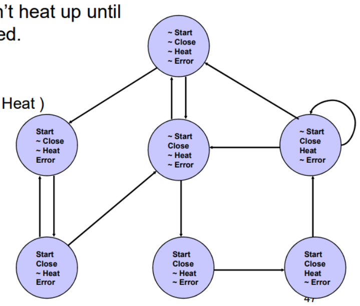
LTL Model Checking Example

- The oven doesn't heat up until the door is closed.

(¬Heat) **U** Close

(¬Heat) **W** Close

G (not Closed => not Heat)



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.

SPIN: The Promela Language

- PROcess MEta LAnguage
- Asynchronous composition of independent processes
- Communication using channels and global variables
- Non-deterministic choices and interleavings

Mutual Exclusion in SPIN

```
bool turn;
bool flag[2];
proctype mutex0() {
again:
  flag[0] = 1;
  turn = 1;
  (flag[1] == 0 || turn == 0);
  /* critical section */
  flag[0] = 0;
  goto again;
}
```