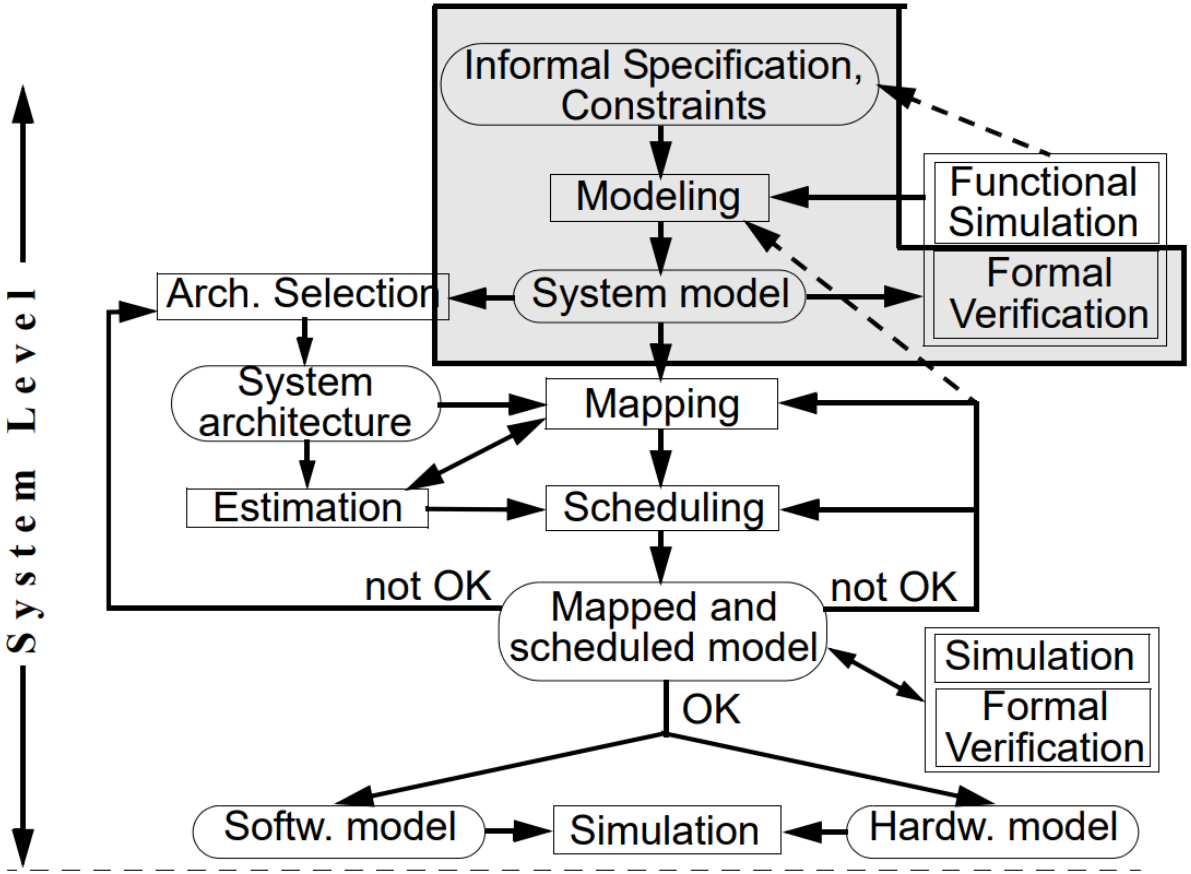
Introduction to Lab 2
Formal Verification with UPPAAL

Reflection on Lab 1

- Deliverables: report & source files (via email – xiaopeng.teng@liu.se)
  - "sensitive": controller → sensors (value change)
    - The value of the sensors: (1, 1, 0, 1) → (1, 1, 0, 1). The control logic in the controller will not be triggered.
    - Use "clk" to synchronize the sensors and the controller
      - 0 → 1 → 0 → 1 → …
  - Report:
    - No specific template
    - Describe your design briefly (modules, channels, the control logic)
    - Present the simulations you have conducted (random & specific inputs)
    - Demonstrate how the properties can be satisfied (e.g., screenshot & essential explanations)

Lab 2

- **Formal verification with UPPAAL.**



Formal Methods

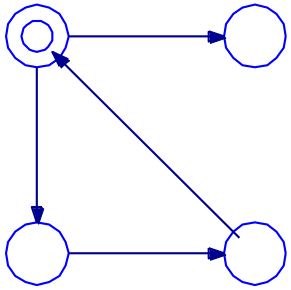
For the levels of complexity typical to embedded systems:

- Traditional validation techniques such as simulation cover only a small fraction of the system's behavior
- Bugs found at late stages of the production pipeline have a negative impact on the time-to-market
- A failure may lead to a catastrophe

Model Checking

System Description

Automata model **N**



Specification (Req. Properties)

Temporal Logic formula **f**

AG $!(p_c \& p_d)$

$\mathbf{EF}_{<2}\ p_e$

Model Checker

$\mathbf{N} \stackrel{?}{\models} \mathbf{f}$

yes

???

no

Diagnostic Information

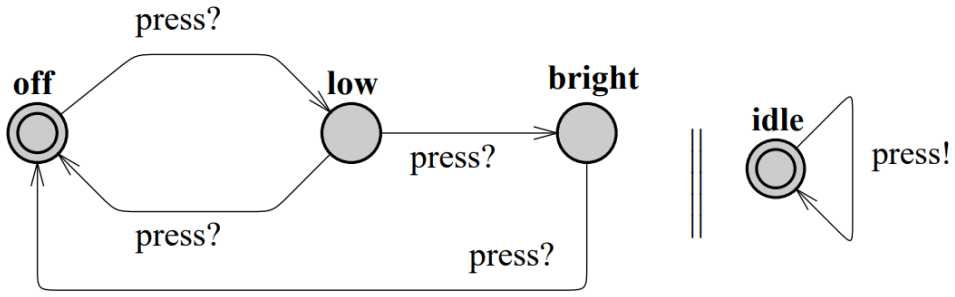
UPPAAL

- Modeling: **timed automata**
- Validation: **simulation**
- Verification: **model checking**
- Requirements: **computation tree logic**

- For more details: http://www.uppaal.org/
  - User-friendly graphical user interface
  - Free for academic use

Timed Automata in UPPAAL

- A timed automaton is a finite automaton augmented with a finite set of real variables called **clocks**
- All the clocks change along the time line with the **same constant rate**
- Timing constraints can be expressed by imposing **conditions** over clocks
- The model consists of a collection of timed automata that operate and coordinate with each other through **shared variables** and **synchronization labels**

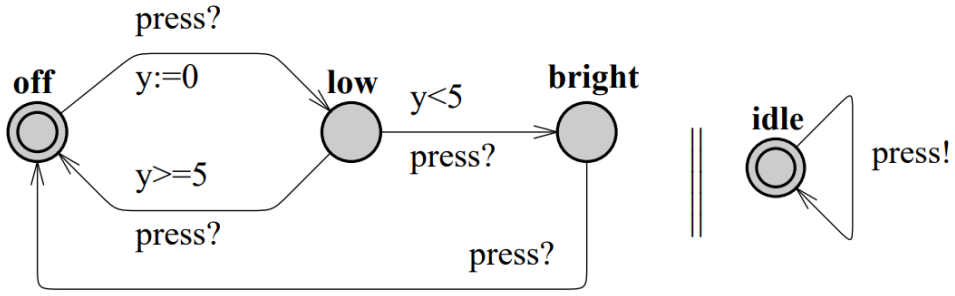


(a) Lamp. (b) User.

Fig. 1. The simple lamp example.

Timed Automata in UPPAAL

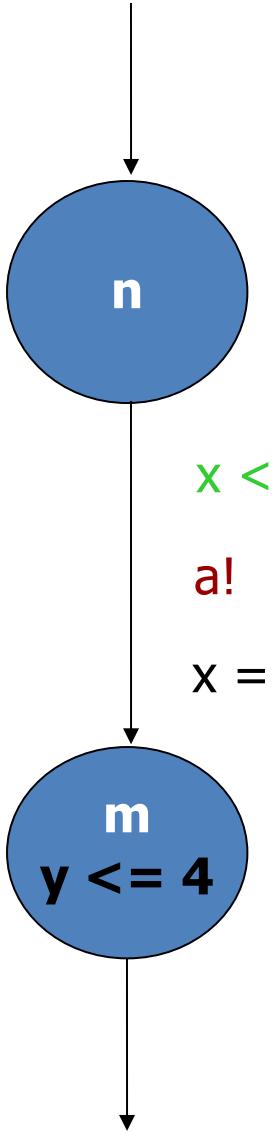
- A timed automaton is a finite automaton augmented with a finite set of real variables called **clocks**
- All the clocks change along the time line with the **same constant rate**
- Timing constraints can be expressed by imposing **conditions** over clocks
- The model consists of a collection of timed automata that operate and coordinate with each other through **shared variables** and **synchronization labels**



(a) Lamp. (b) User.

Fig. 1. The simple lamp example.

Timed Automata: Syntax



States: n, m

Clocks: x, y

Channels: a

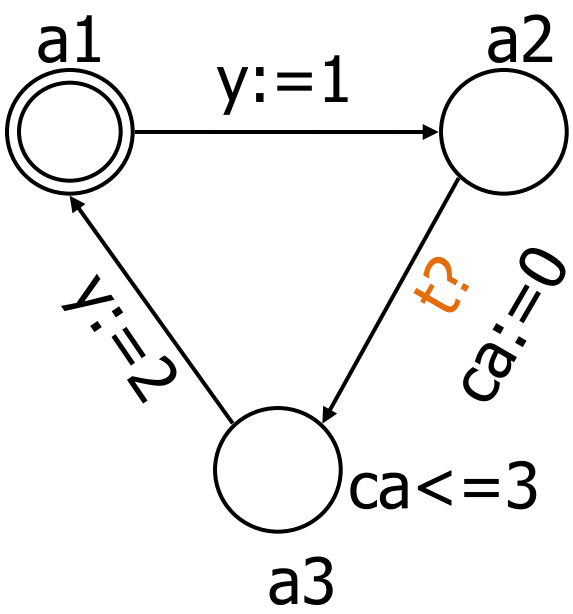
Guards

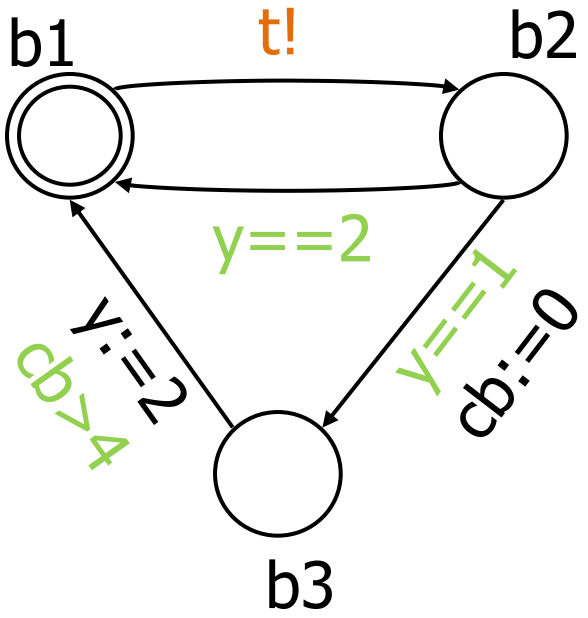
Synchronizations

Updates

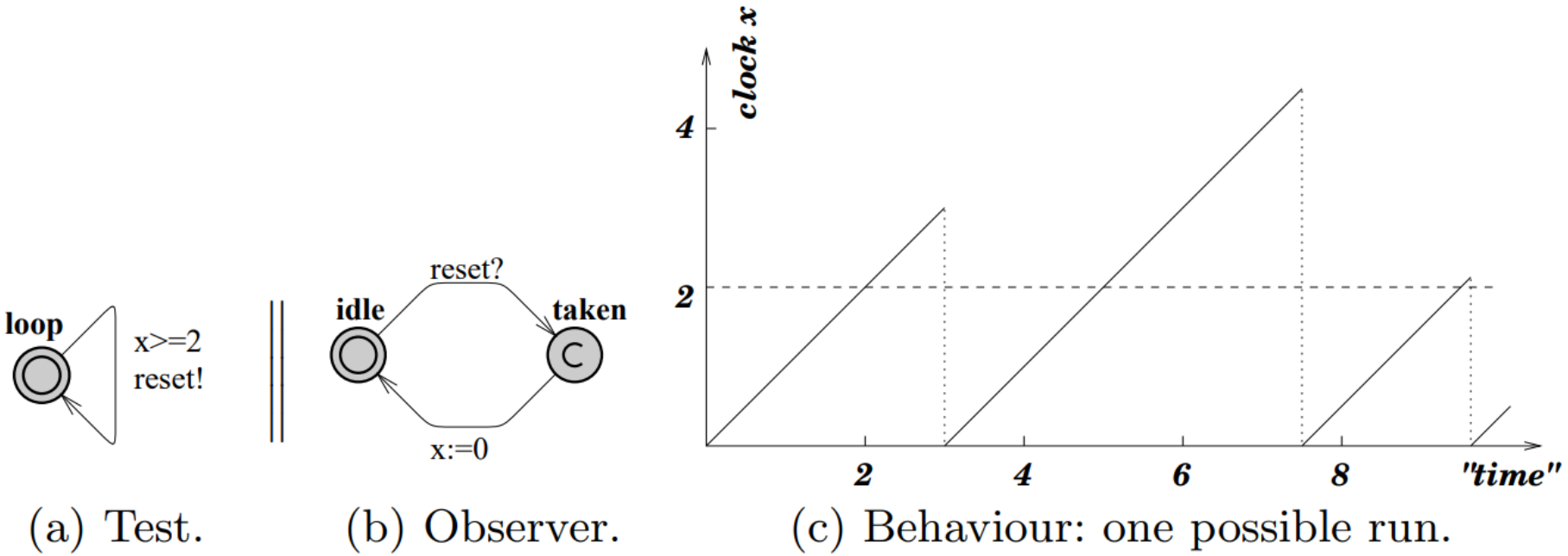
Invariants (progress condition)

Timed Automata: Example





Timed Automata: Example

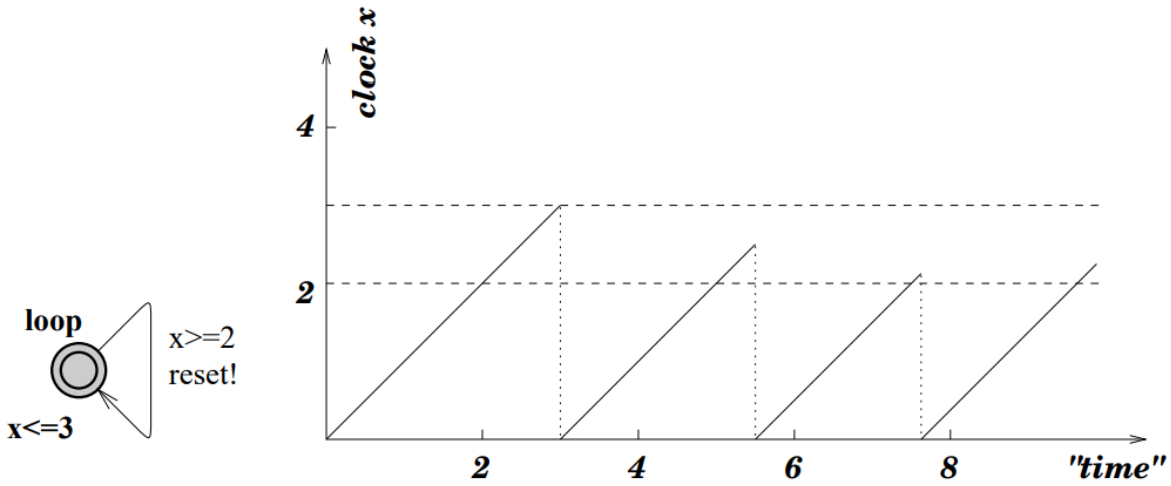


(a) Test. (b) Observer. (c) Behaviour: one possible run.

Fig. 5. First example with an observer.

https://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf

Timed Automata: Example

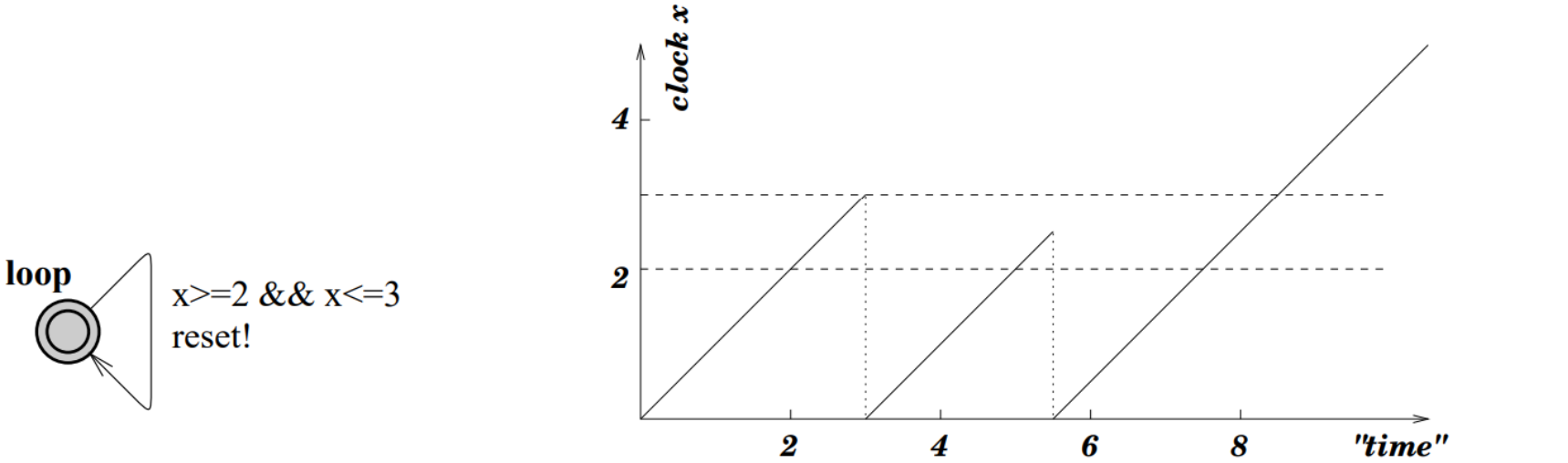


(a) Test. (b) Updated behaviour with an invariant.

Fig. 6. Updated example with an invariant. The observer is the same as in Fig. 5 and is not shown here.

https://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf

Timed Automata: Example



(a) Test. (b) Updated behaviour with a guard and no invariant.

Fig. 7. Updated example with a guard and no invariant.

https://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf

Temporal Logics

How do we specify properties for timed systems?

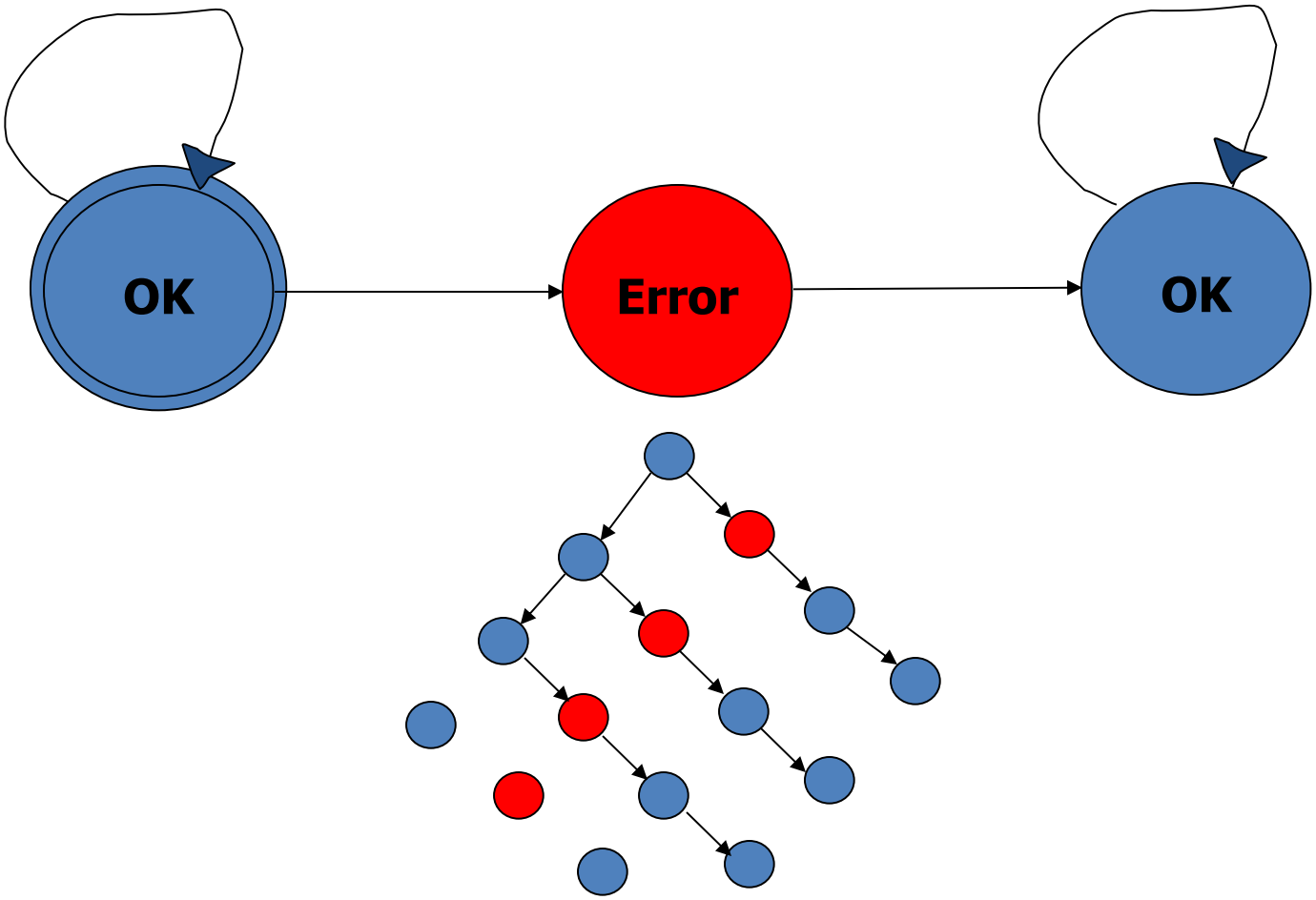
- Logic augmented with temporal modal operators
  - Allow us to reason about how the truth of assertions changes over time
- Used to specify desired properties of timed systems
  - Safety: Nothing bad will ever happen
  - Liveness: Something good may eventually happen
  - Boundedness: Something will happen within a time limit
- Different forms of temporal logic
  - Depending on the underlying model of time
  - Computation tree logic

Computation Tree Logic (CTL)

- Based on propositional logic of branching time
  - The time may split into more than one possible future
- Formulas are composed of atomic propositions (states in timed automata) and boolean connectors
- Temporal operators:
  - Path quantifier
    - A – all computation paths
    - E – some computation path
  - Forward-time operators
    - G – globally
    - F – in the future
    - X – next time
    - U – until

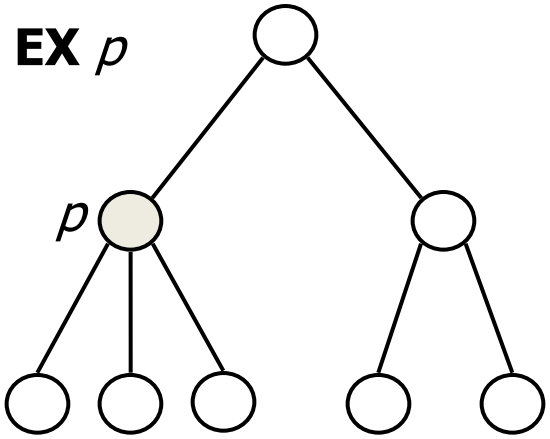
Computation Tree

- Represents an unfolded state graph – nodes are the possible states that the system might reach
- Build (infinite) computation trees from the TA model

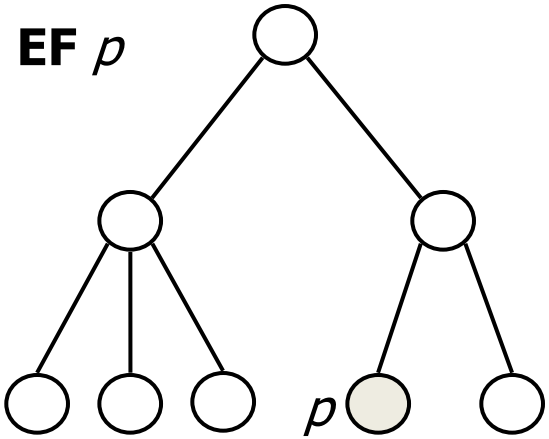


CTL: Temporal Operators

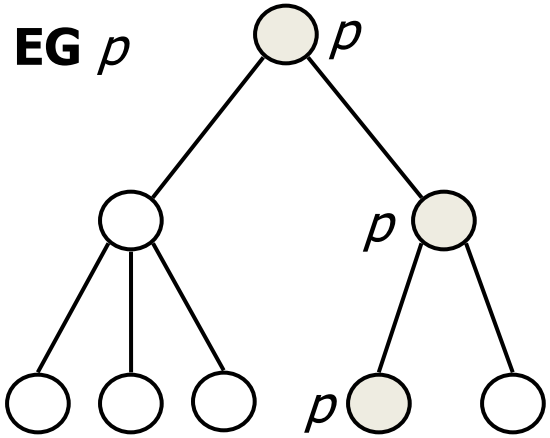
Potentially next



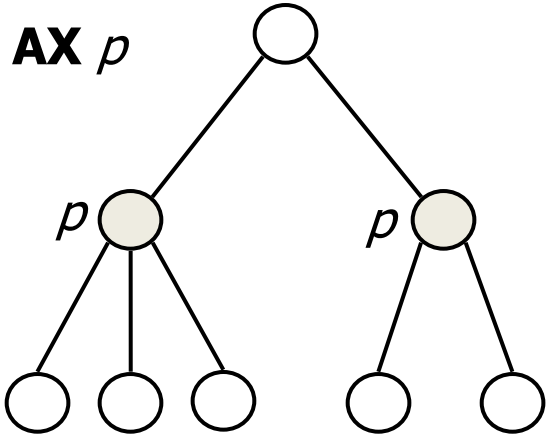
Potentially



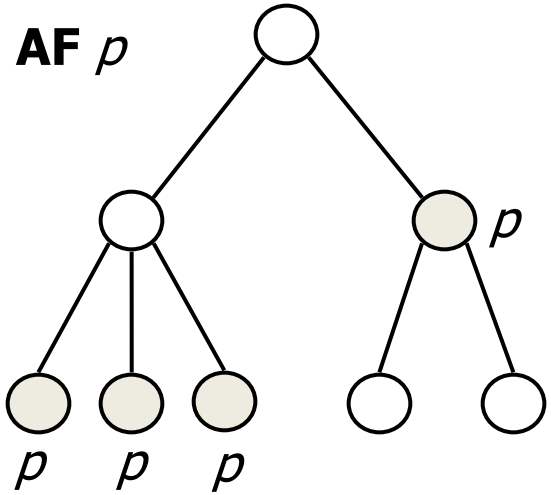
Potentially always



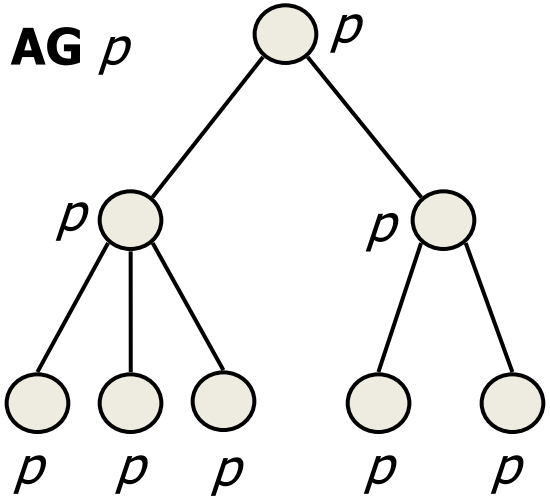
Inevitably next



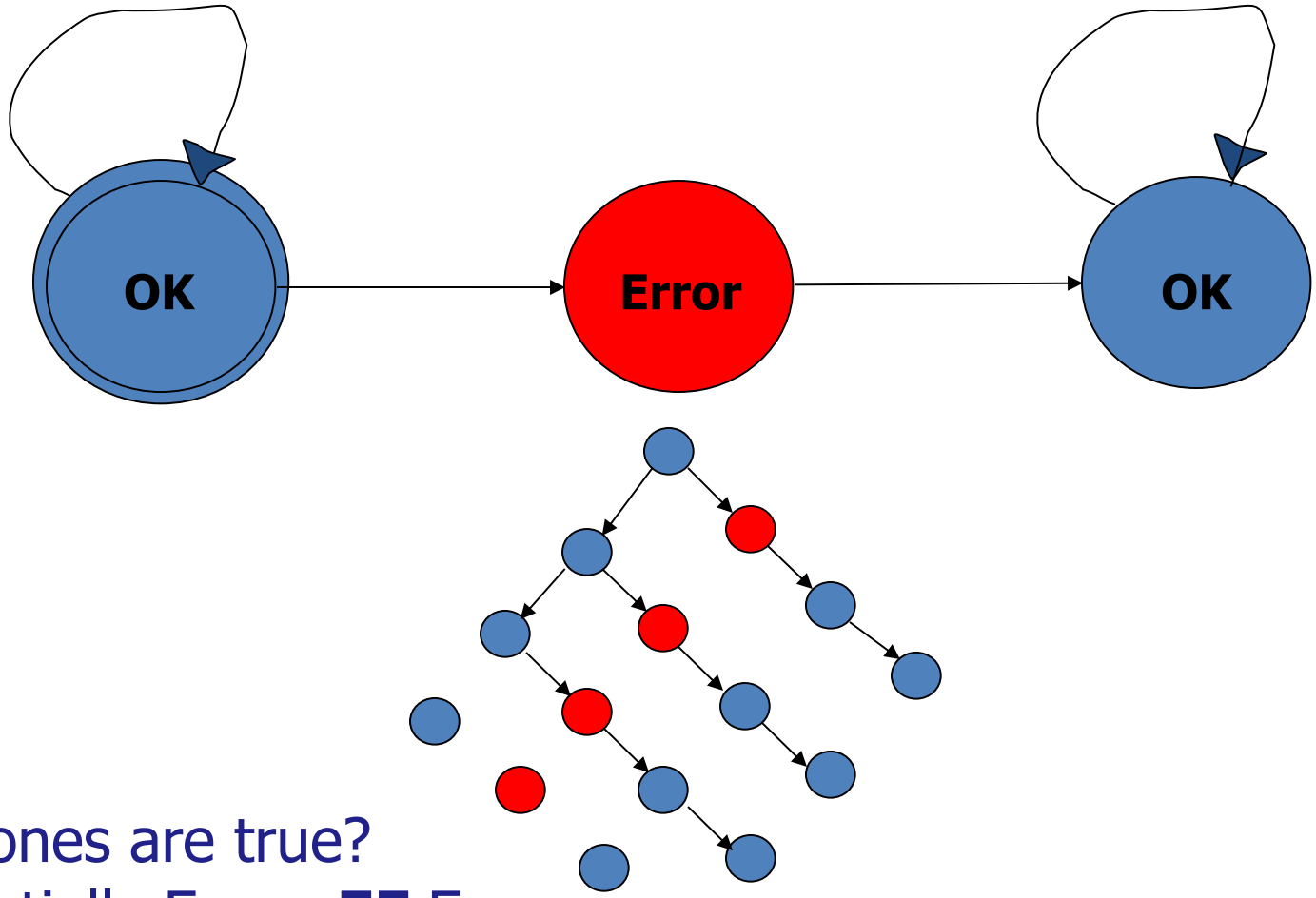
Inevitably



Inevitably always



CTL: Example



Which ones are true?

- Potentially Error: **EF** Error
- Potentially always OK: **EG** OK
- Inevitably always OK: **AG** OK

CTL in UPPAAL

- UPPAAL has a special syntax for CTL
  - AF *p* = A<> *p* — *Inevitably p*
  - AG *p* = A[] *p* — *Inevitably always p*
  - EF *p* = E<> *p* — *Potentially p*
  - EG *p* = E[] *p* — *Potentially always p*
  - Boolean connectives: and, or, not, imply
  - No nested formulas, except
    - **A[]** (*p* imply **A<>** *q*) = *p* --> *q*
    - If *p* holds in some state, *q* will hold in some future state.
    - Can be used to verify one of the properties in the assignments: ”If a car arrives at the traffic light, it should eventually be granted the green light.”

Formal Methods

Formal methods can:

- Overcome some of the limitations of traditional techniques
- Give a better understanding of the system
- Help to uncover ambiguities
- Reveal new insights of the system

Formal methods do have limitations and are to complement simulation and testing rather than to replace them.

Lab Assignment

- Study the lab material linked from the web pages
- There you will find the lab assignments as well as the requirements on the deliverables
  - Introductory assignments to get familiar with timed automata, CTL, and UPPAAL
  - Model the traffic light controller in timed automata
    - Simulate and verify
  - Implement a communication protocol
    - Alternating bit protocol

Demonstration for Lab 2
Formal Verification with UPPAAL

The Gossiping People

There are n people. Each has a secret. They are desperate to tell their secrets to each other. They communicate over the phone. When two people are on the phone, they exchange the secrets they currently know. What is the minimum number of calls needed so that each person knows all secrets?

- How to model phone calls? Channels.
- How to represent secrets? Bit flags.
- How to exchange secrets? Global variables.

Hints for Lab 2

- Read "short UPPAAL tutorial"
  - Concepts:
    - Synchronization
    - Clock/Time
    - Variables
    - Invariant
    - Urgent/committed locations
    - Broadcast channel

- Assignment 3.4: traffic light controller
  - Less complicated than the "systemC" design
  - Refer to the "Fisher" example (mutex property)

Thank you!
Questions?