EurAi Fellow

AAIA Fellow

Formal Aspects of Strategic Reasoning and Game Playing

Munyque Mittelmann, **Aniello Murano**, Laurent Perrussel

University of Naples Federico II
University Toulouse Capitole

ESSAI 2024
Athens Greece
July 2024

INdAM

PRIN 2020

FAIR Future Artificial Intelligence Research

ASTREA

https://people.na.infn.it/~murano/

Outline

Day 3

1.1 Basic concepts of formal verification for monolithic systems (45 slides/45 min)

- Introduction to closed system verification: **Model Checking**
- Linear and Branching-time Temporal Logics: **LTL**, **CTL**, and **CTL***
- An **automata-theoretic approach** to solve model checking

1.2 From one player to two players (30 sides/30 min)

- Introduction to open systems verification: **Module checking** as a two-player game

Day 4

2.1 From two-players to multiple players (75 slides/75 min)

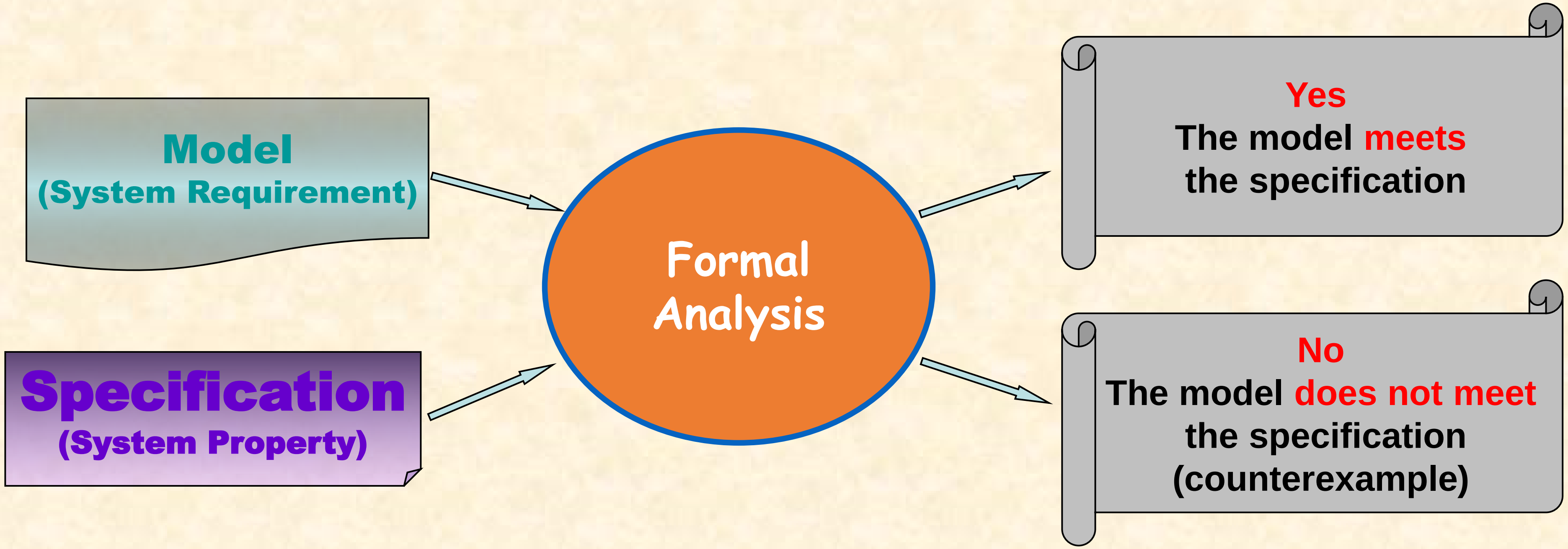
- Logics for strategic reasoning: ATL and ATL*
- An automata-theoretic approach and a fixed-point algorithm to solve model checking
- **From ATL to Strategy Logic**

Preface: System Correctness

- Hardware and software systems are growing up in their abilities and applications.
- From health-care and transportation to smartphones, systems are becoming more and more complex and intelligent!
- System failure can affect safety and induces a lost of money, as well as time and market reputation.
- A notable example: Pentium IV bag: 4195835 – 4195835 / 3145727 * 3145727, doesn’t return 0, but 256. It costed $500 million.
- System failure is not an option!!!

Preface: A Solution Approach

- Formal verification:
  - We can check whether a system is correct with respect to a desired behavior (specification), by formally checking whether a representation of the system meets the specification.

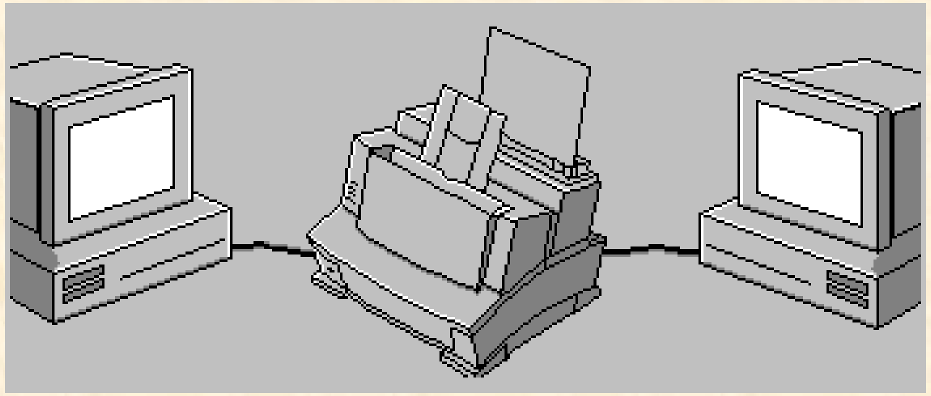


Advantages of Formal Methods

- Apply to system models
- Used at a very early stage of a project
- Based on robust mathematical theories
- Exhaustive as they can check all possible computations
- Diagnostic counterexamples
- No problem with partial specifications
- Several existing tools!

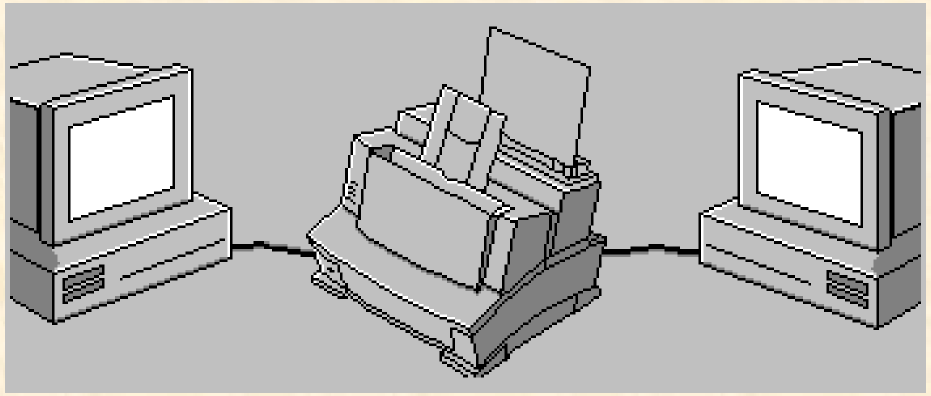
Example: Scheduler

- A scheduler should be designed so that jobs of the two users **are not printed simultaneously**, and whenever a user sends a job, the job is printed **eventually**.



Example: Scheduler

- A scheduler should be designed so that jobs of the two users **are not printed simultaneously**, and whenever a user sends a job, the job is printed **eventually**.



- Using formal methods, we can check reliability for such a scheduler by:
  - Providing an appropriate model for the scheduler **M**
  - A specification for the desired behavior **ϕ**
  - A formal technique that allows to check that **M meets ϕ**

System Verification Scenarios

- The **model** and **specification** framework depend on the specific system and behavior we are dealing with.

- The **decision problem** (algorithm analysis) also depends on the specific setting we are facing.

Possible System Scenarios

Possible System Scenarios

- Closed systems:

- Open (system vs. environment) systems:

- Multi-agent systems:

Possible System Scenarios

- Closed systems:
  - Behavior is fully characterized by system states (one source of nondeterminism).

- Open (system vs. environment) systems:

- Multi-agent systems:

Possible System Scenarios

- Closed systems:
  - Behavior is fully characterized by system states (one source of nondeterminism).

- Open (system vs. environment) systems:
  - Interaction with an unpredictable environment (two source of non-determinism)

- Multi-agent systems:

Possible System Scenarios

- Closed systems:
  - Behavior is fully characterized by system states (one source of nondeterminism).
- Open (system vs. environment) systems:
  - Interaction with an unpredictable environment (two source of non-determinism)
- Multi-agent systems:
  - The system is composed of several entities acting adversarial or in a cooperative way.

Possible Specification Formalisms

- Temporal logics:
  - Linear such as LTL
  - Branching such as CTL, and CTL*

- Multi-agent temporal logics:
  - Alternating-time temporal logic (ATL)
  - Strategy Logic (SL)

System Analysis

- Decision problems:
  - Model Checking
  - Satisfiability
  - Module Checking/Games
  - Reactive Synthesis

Part 1.1

- ✓ Introduction to formal verification;
- → **Models for closed systems: Kripke Structures;**
- ➢ Linear and branching-time temporal logics: LTL, CTL, and CTL*;
- ➢ Decision problems: model checking and satisfiability.
- ➢ Automata on infinite words and trees.

A Basic Model: Kripke Structure

- Systems can be represented as labeled-state transition graphs: **Kripke Structures**
- Formally,

$$\mathbf{M= (AP, S, S_0, R, Lab)}$$

- AP is a set of atomic propositions
- S is a finite set of states
- $S_0 \subseteq S$ is the set of initial states
- $R \subseteq S \times S$ is a transition relation, total: $\forall s \in S, \exists s' . R(s, s')$
- $Lab : S \rightarrow 2^{AP}$ labels each state with propositions true in the state

Kripke Structure Applications

- Kripke structures are suitable to model basic system behaviors in a very natural way.
- They are very efficient in modelling controllers:

Kripke Structure Applications

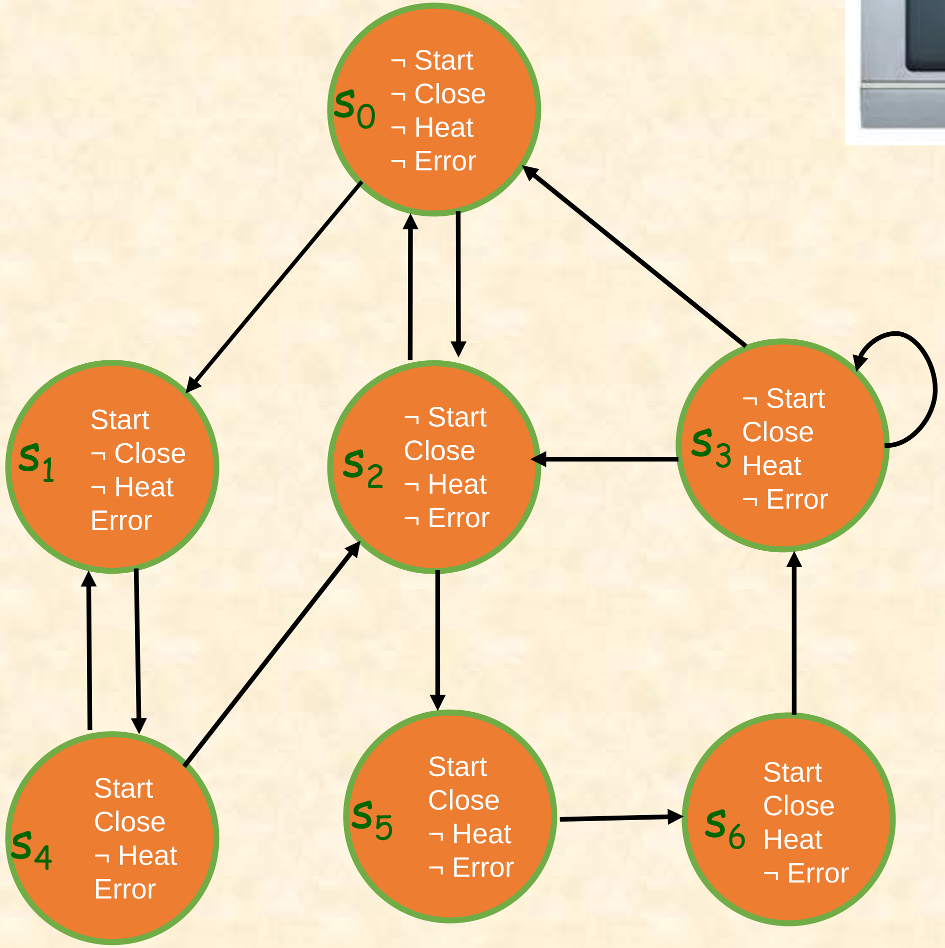
- Kripke structures are suitable to model basic system behaviors in a very natural way.
- They are very efficient in modelling controllers:
  - In a traffic-light system, we can model: “if the light was red at the previous state and is orange now, it must turn green at the next state”.

Kripke Structure Applications

- Kripke structures are suitable to model basic system behaviors in a very natural way.
- They are very efficient in modelling controllers:
  - In a traffic-light system, we can model: "if the light was red at the previous state and is orange now, it must turn green at the next state".
  - In a train system , we can model: "If a train is entering the tunnel now, the semaphore has been switched red on the other side at the previous moment".

A concrete example: Microwave Oven

- AP = {Start, Close, Heat, Error}
- $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$
- $S_0 = \{s_0\}$
- R and Lab are as in the figure



Part 1.1

- ✓ Introduction to formal verification;
- ✓ Models for closed systems: Kripke Structures;
- → **Linear and branching-time temporal logics: LTL, CTL and CTL***
- ➢ Decision problems: model checking and satisfiability.
- ➢ Automata on infinite words and trees.

Temporal Logic Specification

- Temporal logics allows to describe the evolution of system along the time.
  - We intrinsically assume that system computations are infinite.

- Temporal logics extend classical proposition logic with temporal operators.

- Depending on the underling nature of the time, we distinguish between:

  - **Linear-time temporal-logics**

  - **Branching-time temporal-logics**

Temporal Logic Specification

- Temporal logics allows to describe the evolution of system along the time.
  - We intrinsically assume that system computations are infinite.

- Temporal logics extend classical proposition logic with temporal operators.

- Depending on the underling nature of the time, we distinguish between:

  - **Linear-time temporal-logics**
    - Every moment has a unique successor
    - Infinite sequences (words)

  - **Branching-time temporal-logics**

Temporal Logic Specification

- Temporal logics allows to describe the evolution of system along the time.
  - We intrinsically assume that system computations are infinite.

- Temporal logics extend classical proposition logic with temporal operators.

- Depending on the underling nature of the time, we distinguish between:

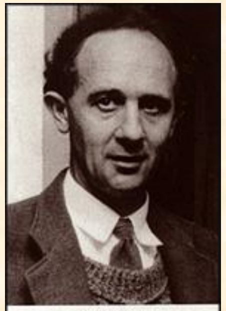
  - **Linear-time temporal-logics**
    - Every moment has a unique successor
    - Infinite sequences (words)

  - **Branching-time temporal-logics**
    - Every moment has several successors
    - Infinite trees

History of Temporal Logic and Formal Verification

History of Temporal Logic and Formal Verification

- Temporal logic begat as a philosophical study: ethics, free will, etc. **Arthur Prior** in the '50 is the first to use a concept of time-delay in computer circuits. With his **Tense Logic**, Prior has inspired many researcher.



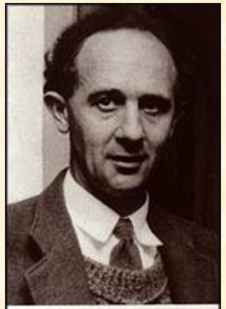
Arthur Prior
1914–1969

History of Temporal Logic and Formal Verification

- Temporal logic begat as a philosophical study: ethics, free will, etc. **Arthur Prior** in the '50 is the first to use a concept of time-delay in computer circuits. With his **Tense Logic**, Prior has inspired many researcher.
- In 1977, **Amir Pnueli** is the first to use a future linear temporal logic (LTL) for the specification of non-terminating and concurrent programs: A temporal logic with "next" and "until".

History of Temporal Logic and Formal Verification

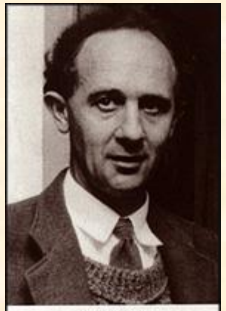
- Temporal logic begat as a philosophical study: ethics, free will, etc. **Arthur Prior** in the ‘50 is the first to use a concept of time-delay in computer circuits. With his **Tense Logic**, Prior has inspired many researcher.
- In 1977, **Amir Pnueli** is the first to use a future linear temporal logic (LTL) for the specification of non-terminating and concurrent programs: A temporal logic with “next” and “until”.
- Edmund Clarke and Ernest Allen Emerson in the early 1980’s developed a framework to temporal logic reasoning about programs (**CTL and Model Checking**);
- Independently, Jean-Pierre Queille and Joseph Sifakis essentially proposed the same method at this time.



Arthur Prior
1914–1969

History of Temporal Logic and Formal Verification

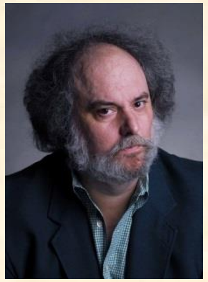
- Temporal logic begat as a philosophical study: ethics, free will, etc. **Arthur Prior** in the '50 is the first to use a concept of time-delay in computer circuits. With his **Tense Logic**, Prior has inspired many researcher.
- In 1977, **Amir Pnueli** is the first to use a future linear temporal logic (LTL) for the specification of non-terminating and concurrent programs: A temporal logic with "next" and "until".
- Edmund Clarke and Ernest Allen Emerson in the early 1980's developed a framework to temporal logic reasoning about programs (**CTL and Model Checking**);
- Independently, Jean-Pierre Queille and Joseph Sifakis essentially proposed the same method at this time.
- Pnueli won the 1996 Turing award for his contribution to temporal logic specifications



Arthur Prior
1914–1969

History of Temporal Logic and Formal Verification

- Temporal logic begat as a philosophical study: ethics, free will, etc. **Arthur Prior** in the '50 is the first to use a concept of time-delay in computer circuits. With his **Tense Logic**, Prior has inspired many researcher.
- In 1977, **Amir Pnueli** is the first to use a future linear temporal logic (LTL) for the specification of non-terminating and concurrent programs: A temporal logic with "next" and "until".
- Edmund Clarke and Ernest Allen Emerson in the early 1980's developed a framework to temporal logic reasoning about programs (**CTL and Model Checking**);
- Independently, Jean-Pierre Queille and Joseph Sifakis essentially proposed the same method at this time.
- Pnueli won the 1996 Turing award for his contribution to temporal logic specifications
- Clarke, Emerson, and Sifakis won the 2007 Turing award for their contribution to Model Checking



LTL: Linear-Time Temporal-Logic [Pnueli' 77]

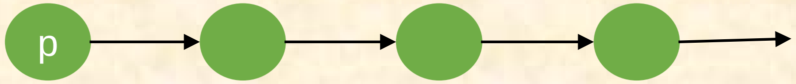
- Determines patterns on infinite traces $\pi = s_0 s_1 s_2 \ldots$.
- Elements:
  - Atomic Propositions: AP
  - Boolean Operations: {¬,∨,∧}
  - Temporal operators: {X, F, G, U}

p "p is true now" (p ∈ AP)

LTL: Linear-Time Temporal-Logic [Pnueli' 77]

- Determines patterns on infinite traces $\pi = s_0 s_1 s_2 \ldots$
- Elements:
  - Atomic Propositions: AP
  - Boolean Operations: $\{\neg, \vee, \wedge\}$
  - Temporal operators: {X, F, G, U}

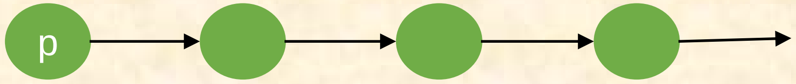
p "p is true now" ($p \in AP$)



LTL: Linear-Time Temporal-Logic [Pnueli' 77]

- Determines patterns on infinite traces $\pi = s_0 s_1 s_2 \ldots$
- Elements:
  - Atomic Propositions: AP
  - Boolean Operations: $\{\neg, \vee, \wedge\}$
  - Temporal operators: {X, F, G, U}

p "p is true now" ($p \in AP$)



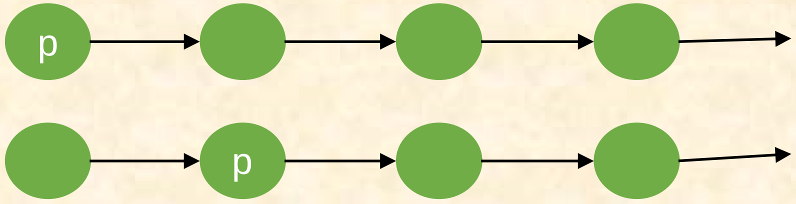
X p "p is true in the neXt state"

LTL: Linear-Time Temporal-Logic [Pnueli' 77]

- Determines patterns on infinite traces $\pi = s_0 s_1 s_2 \ldots$
- Elements:
  - Atomic Propositions: AP
  - Boolean Operations: {¬,∨,∧}
  - Temporal operators: {X, F, G, U}

p "p is true now" (p ∈ AP)

X p "p is true in the neXt state"



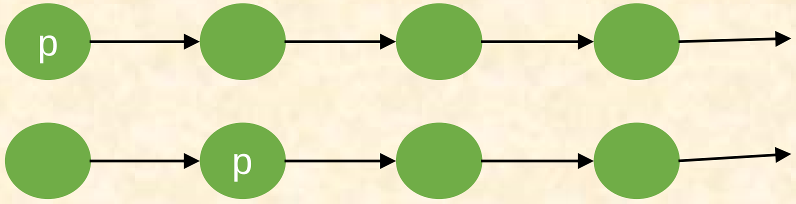
LTL: Linear-Time Temporal-Logic [Pnueli' 77]

- Determines patterns on infinite traces $\pi = s_0 s_1 s_2 \ldots$
- Elements:
  - Atomic Propositions: AP
  - Boolean Operations: {¬,V,Λ}
  - Temporal operators: {X, F, G, U}

p "p is true now" (p ∈ AP)

X p "p is true in the neXt state"

Fp "p will be true in the Future"



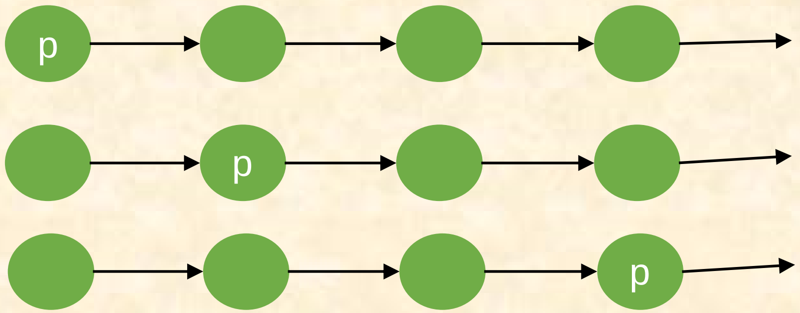
LTL: Linear-Time Temporal-Logic [Pnueli' 77]

- Determines patterns on infinite traces $\pi = s_0 s_1 s_2 \ldots$.
- Elements:
  - Atomic Propositions: AP
  - Boolean Operations: {¬,∨,∧}
  - Temporal operators: {X, F, G, U}

p “p is true now” (p ∈ AP)

X p “p is true in the neXt state”

Fp “p will be true in the Future”



LTL: Linear-Time Temporal-Logic [Pnueli' 77]

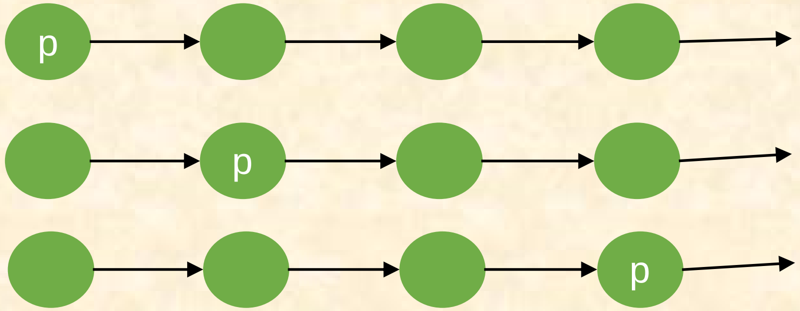
- Determines patterns on infinite traces $\pi = s_0 s_1 s_2 \ldots$.
- Elements:
  - Atomic Propositions: AP
  - Boolean Operations: {¬,∨,∧}
  - Temporal operators: {X, F, G, U}

p "p is true now" (p ∈ AP)

X p "p is true in the neXt state"

Fp "p will be true in the Future"

Gp "p will be Globally true in the future"



LTL: Linear-Time Temporal-Logic [Pnueli' 77]

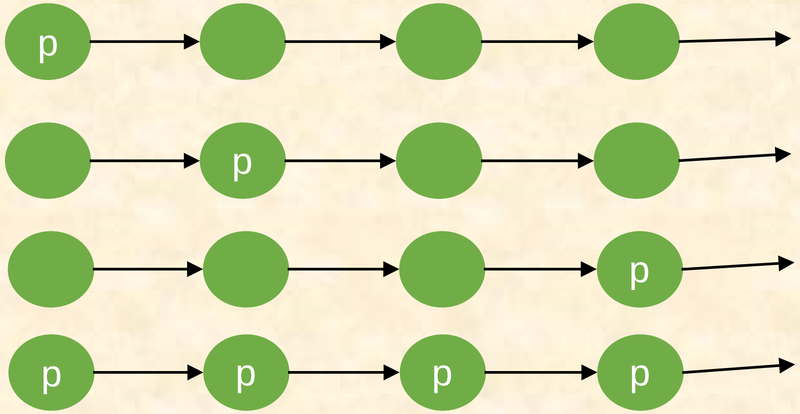
- Determines patterns on infinite traces $\pi = s_0 s_1 s_2 \ldots$
- Elements:
  - Atomic Propositions: AP
  - Boolean Operations: {¬,∨,∧}
  - Temporal operators: {X, F, G, U}

p “p is true now” (p ∈ AP)

X p “p is true in the neXt state”

Fp “p will be true in the Future”

Gp “p will be Globally true in the future”



LTL: Linear-Time Temporal-Logic [Pnueli' 77]

- Determines patterns on infinite traces $\pi = s_0 s_1 s_2 \ldots$
- Elements:
  - Atomic Propositions: AP
  - Boolean Operations: {¬,V,Λ}
  - Temporal operators: {X, F, G, U}

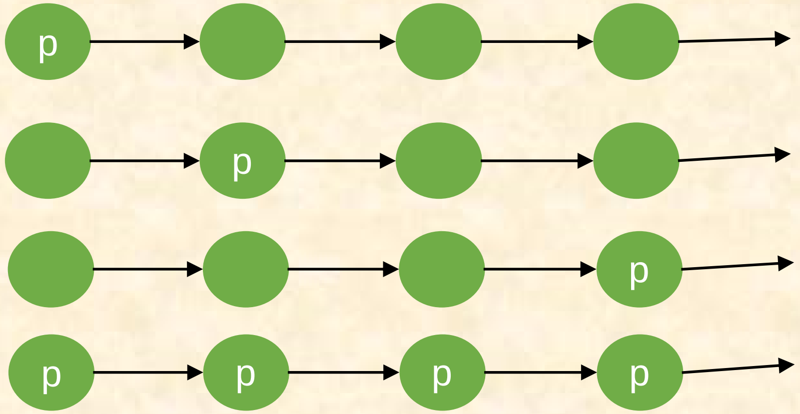
p "p is true now" (p ∈ AP)

X p "p is true in the neXt state"

Fp "p will be true in the Future"

Gp "p will be Globally true in the future"

pUq "p will hold true Until q becomes true"



LTL: Linear-Time Temporal-Logic [Pnueli' 77]

- Determines patterns on infinite traces $\pi = s_0 s_1 s_2 \ldots$.
- Elements:
  - Atomic Propositions: AP
  - Boolean Operations: {¬,∨,∧}
  - Temporal operators: {X, F, G, U}

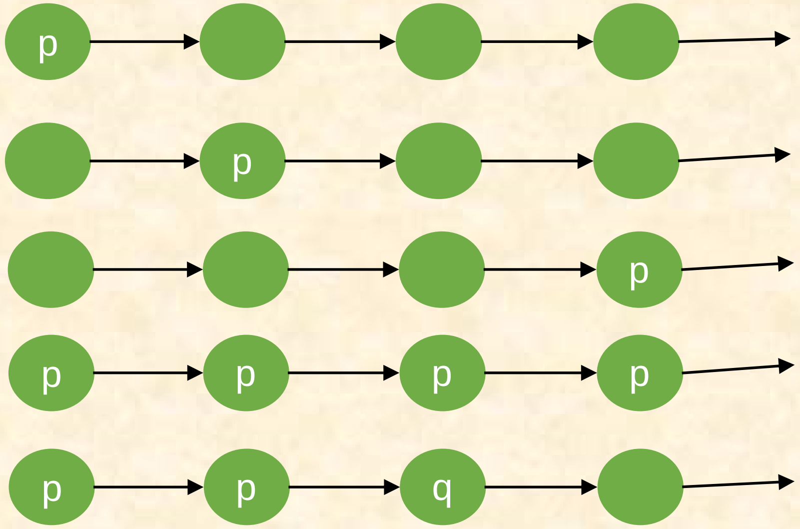
p “p is true now” (p ∈ AP)

X p “p is true in the neXt state”

Fp “p will be true in the Future”

Gp “p will be Globally true in the future”

pUq “p will hold true Until q becomes true”



LTL: Linear-Time Temporal-Logic [Pnueli' 77]

- Determines patterns on infinite traces $\pi = s_0 s_1 s_2 \ldots$
- Elements:
  - Atomic Propositions: AP
  - Boolean Operations: {¬,∨,∧}
  - Temporal operators: {X, F, G, U}

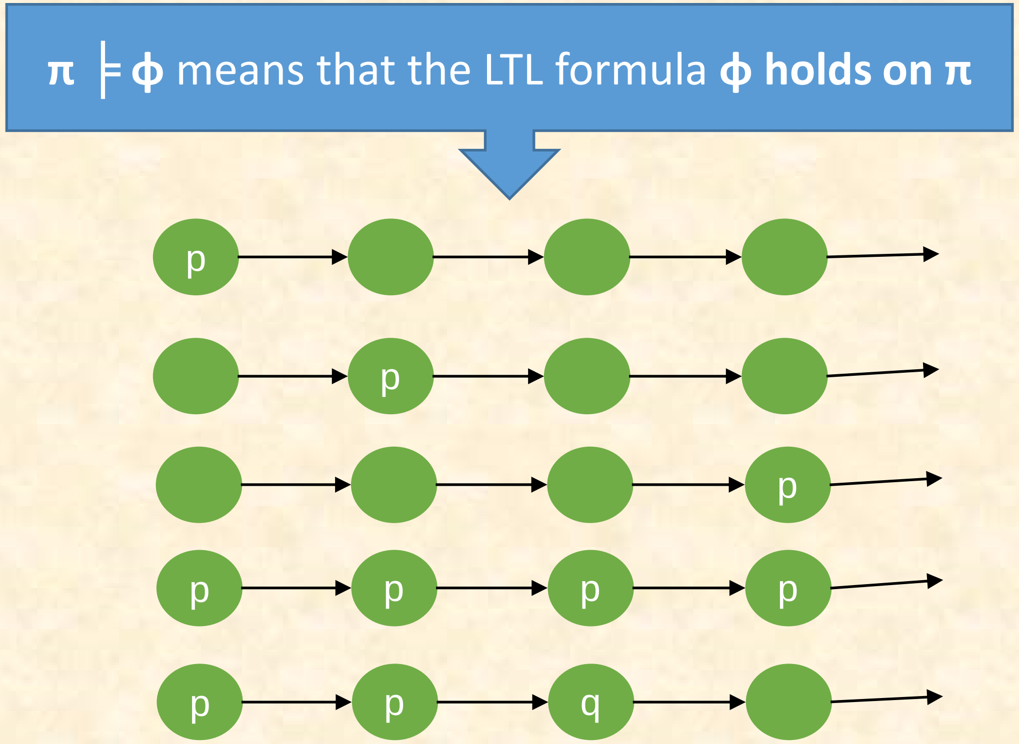
p "p is true now" (p ∈ AP)

X p "p is true in the neXt state"

Fp "p will be true in the Future"

Gp "p will be Globally true in the future"

pUq "p will hold true Until q becomes true"



Example: Safety and Liveness

Example: Safety and Liveness

- Safety: Something bad never happens

Two processes can never be in a critical section at the same time:

$$\neg F(p_1cr \land p_2cr)$$

A process will never meet a critical state:

$$G(\neg error\_state)$$

Example: Safety and Liveness

- Safety: Something bad never happens

Two processes can never be in a critical section at the same time:

$\neg F(p_1 cr \wedge p_2 cr)$

A process will never meet a critical state:

G(¬error_state)

- Liveness: Something desired will happen

Always, every print request is eventually granted:

$G(req \rightarrow F\ grant)$

The microwave doesn’t **heat up** until the **door is closed:**

¬heat_up U door_closed

Always, every repeated request is eventually granted.

$G(GF\ req \rightarrow F\ grant)$

LTL Model Checking

- Given,
  - A Kripke structure M = (AP, S, S_0, R, Lab) modelling the system, an initial state $s_0 \in S_0$ and
  - An LTL formula ϕ over AP representing the specification

LTL Model Checking

- Given,
  - A Kripke structure M = (AP, S, S_0, R, Lab) modelling the system, an initial state $s_0 \in S_0$ and
  - An LTL formula ϕ over AP representing the specification

The **LTL model checking problem**

$$M, s_0 \models \phi$$

LTL Model Checking

- Given,
  - A Kripke structure M = (AP, S, S_0, R, Lab) modelling the system, an initial state $s_0 \in S_0$ and
  - An LTL formula ϕ over AP representing the specification

The **LTL model checking problem**

$$M, s_0 \models \phi$$

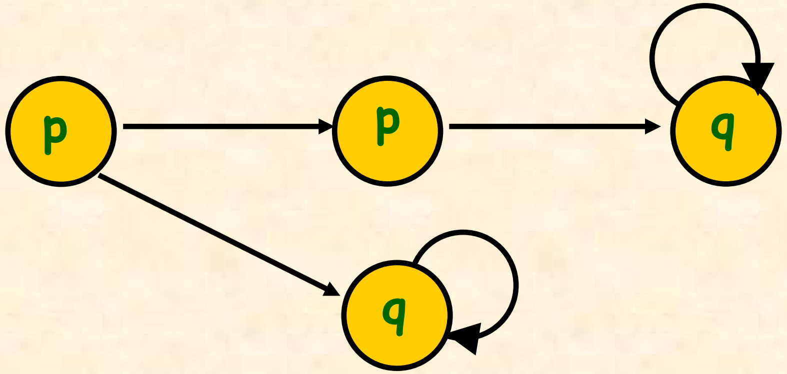
concerns checking whether, **for each path π of M** starting in s_0, we have that $\pi \models \phi$

LTL Satisfiability

- Given an LTL formula ϕ, is there a Kripke structure satisfying the formula?

LTL Satisfiability

- Given an LTL formula ϕ, is there a Kripke structure satisfying the formula?



- Examples:
  - p U q is satisfiable, and the model above is a witness
  - (p U q) ∧ G¬q is not satisfiable

Branching-Time Temporal Logics

- An LTL formula is satisfied over a Kripke structure M if it is satisfied on all its paths
- Paths in M represent all possible system computations
- To restrict the check of a formula to some paths of M, we need a logic that allows to talk about model branches
- To this purpose, we use CTL and CTL*

CTL: Computation Tree Logic

- CTL uses the same temporal operators of LTL.

CTL: Computation Tree Logic

- CTL uses the same temporal operators of LTL.
- Additionally, we use two path quantifiers:
  - **A** means ‘for all computation paths’
  - **E** means ‘there exists a computation path’

AX, AG, AF, AU

EX, EG, EF, EU

CTL: Computation Tree Logic

- CTL uses the same temporal operators of LTL.
- Additionally, we use two path quantifiers:
  - **A** means 'for all computation paths'
  - **E** means 'there exists a computation path'

AX, AG, AF, AU

EX, EG, EF, EU

- In CTL, formulas are evaluated on states rather than paths.

CTL: Computation Tree Logic

- CTL uses the same temporal operators of LTL.
- Additionally, we use two path quantifiers:
  - **A** means ‘for all computation paths’
  - **E** means ‘there exists a computation path’

AX, AG, AF, AU

EX, EG, EF, EU

- In CTL, formulas are evaluated on states rather than paths.
- CTL* allows more complex nesting such as

AXX, EFG, AGX, AFG, EXFG, ...

- CTL* strictly includes both LTL and CTL. Note that LTL and CTL are incomparable

CTL: Computation Tree Logic

- CTL uses the same temporal operators of LTL.
- Additionally, we use two path quantifiers:
  - **A** means 'for all computation paths'
  - **E** means 'there exists a computation path'

AX, AG, AF, AU

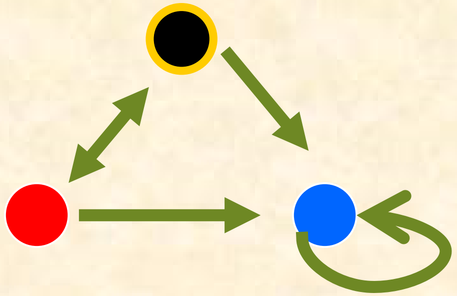
EX, EG, EF, EU

- In CTL, formulas are evaluated on states rather than paths.
- CTL* allows more complex nesting such as

AXX, EFG, AGX, AFG, EXFG, ...

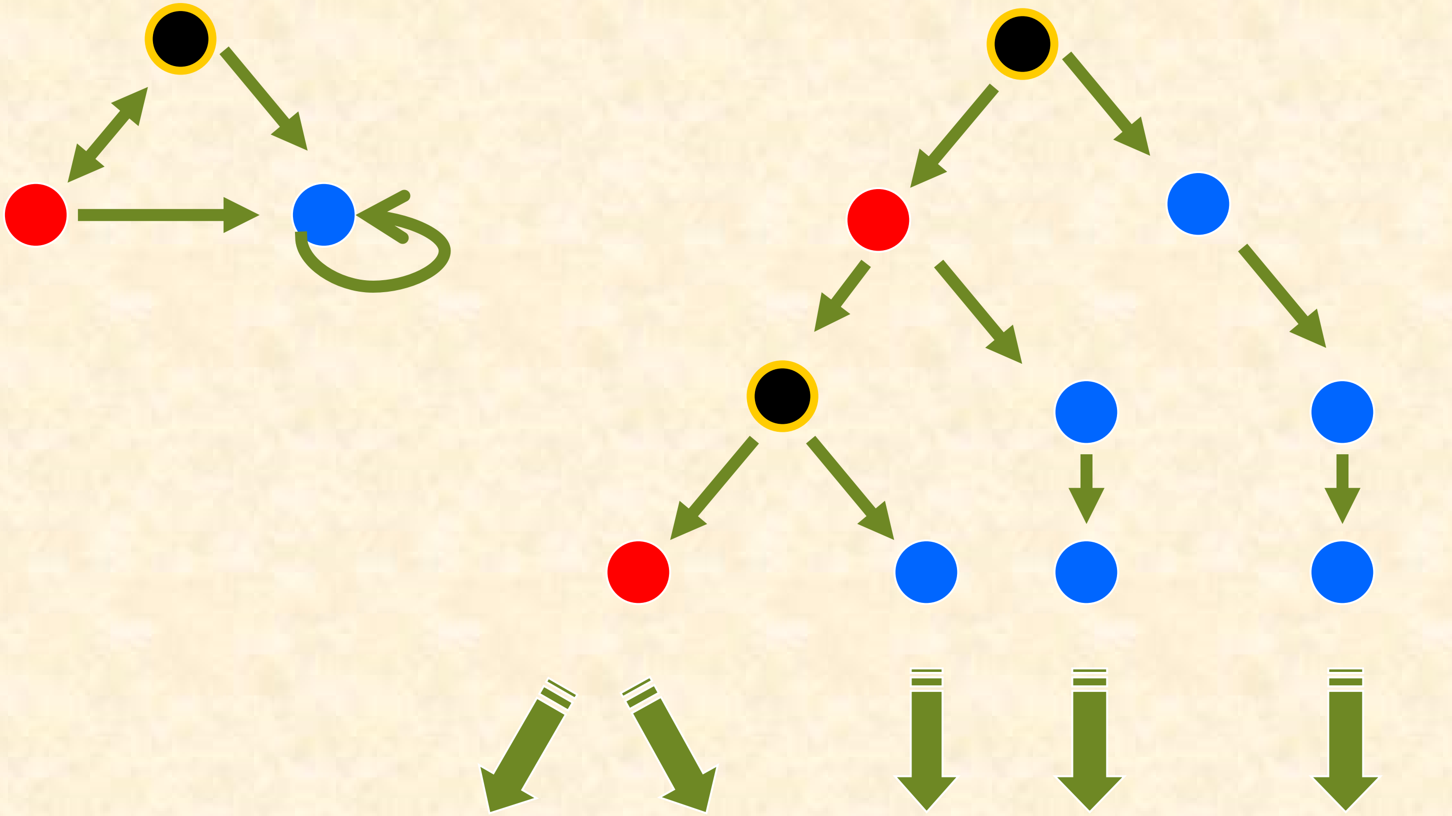
- CTL* strictly includes both LTL and CTL. Note that LTL and CTL are incomparable

Tree model unwinding



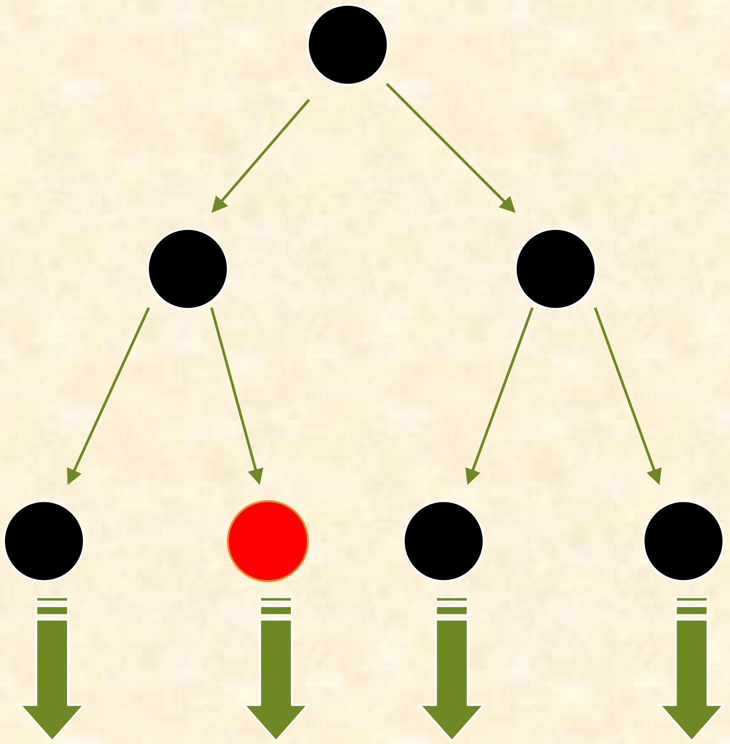
An infinite computation tree

Tree model unwinding



An infinite computation tree

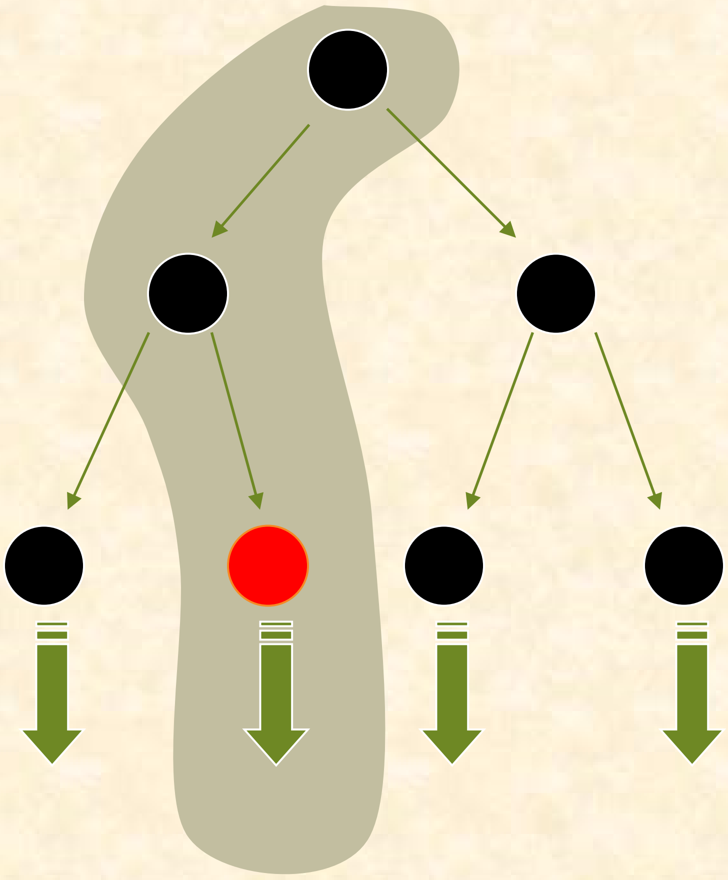
CTL: Computation Tree Logic



EF red

"For at least a path, red will possibly become true"

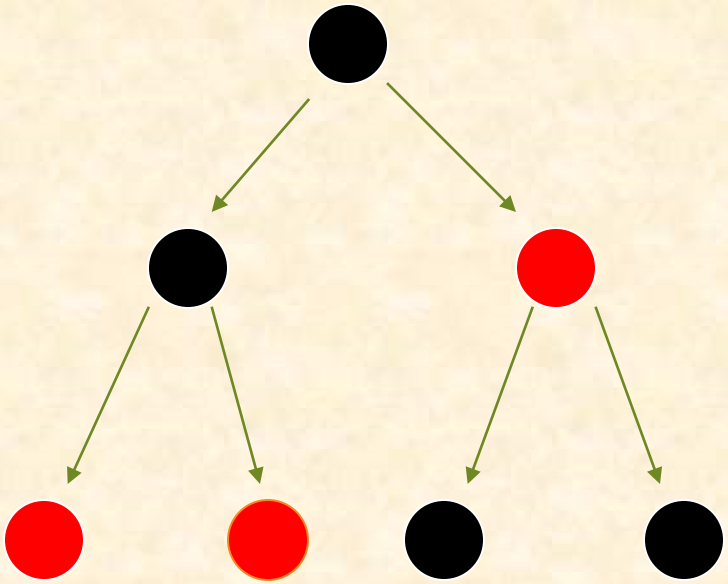
CTL: Computation Tree Logic



EF red

"For at least a path, red will possibly become true"

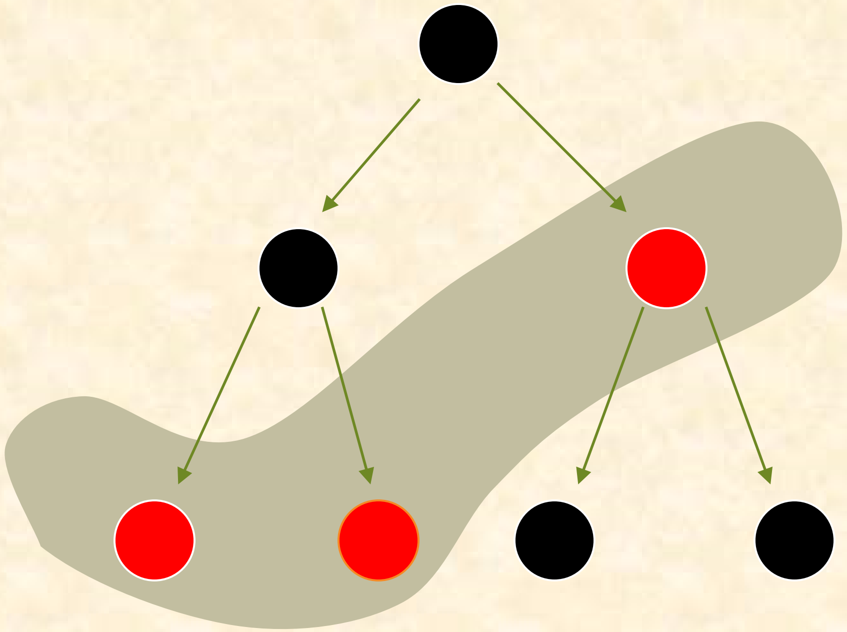
CTL: Computation Tree Logic



AF red

"For every path, red will eventually become true "

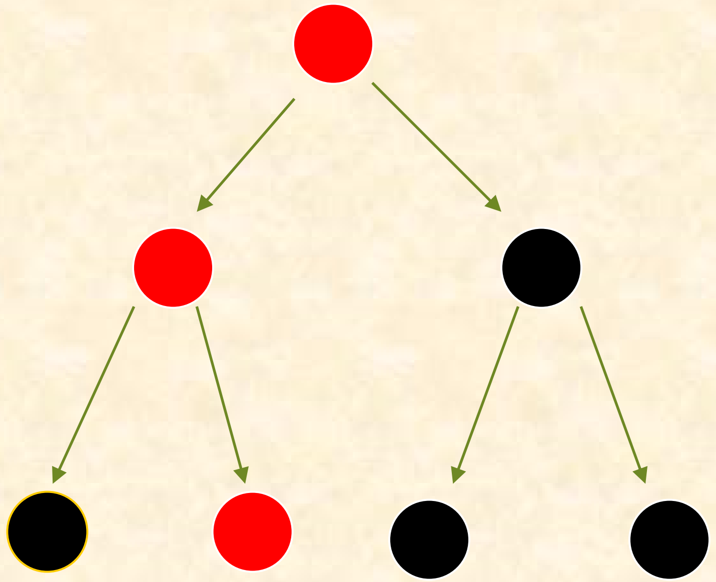
CTL: Computation Tree Logic



AF red

"For every path, red will eventually become true "

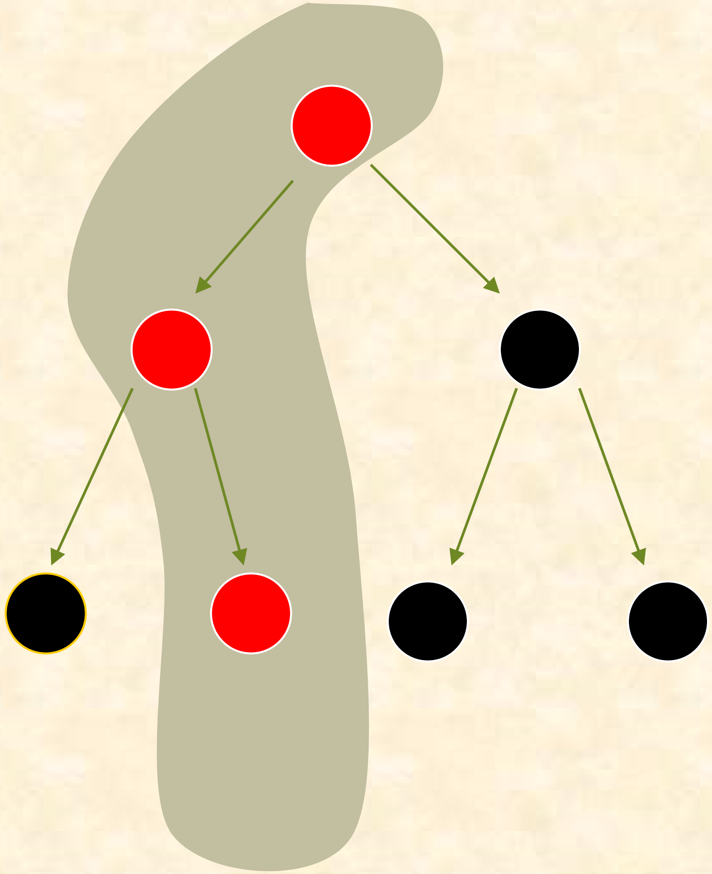
CTL: Computation Tree Logic



EG red

For at least a path, red remains always true”

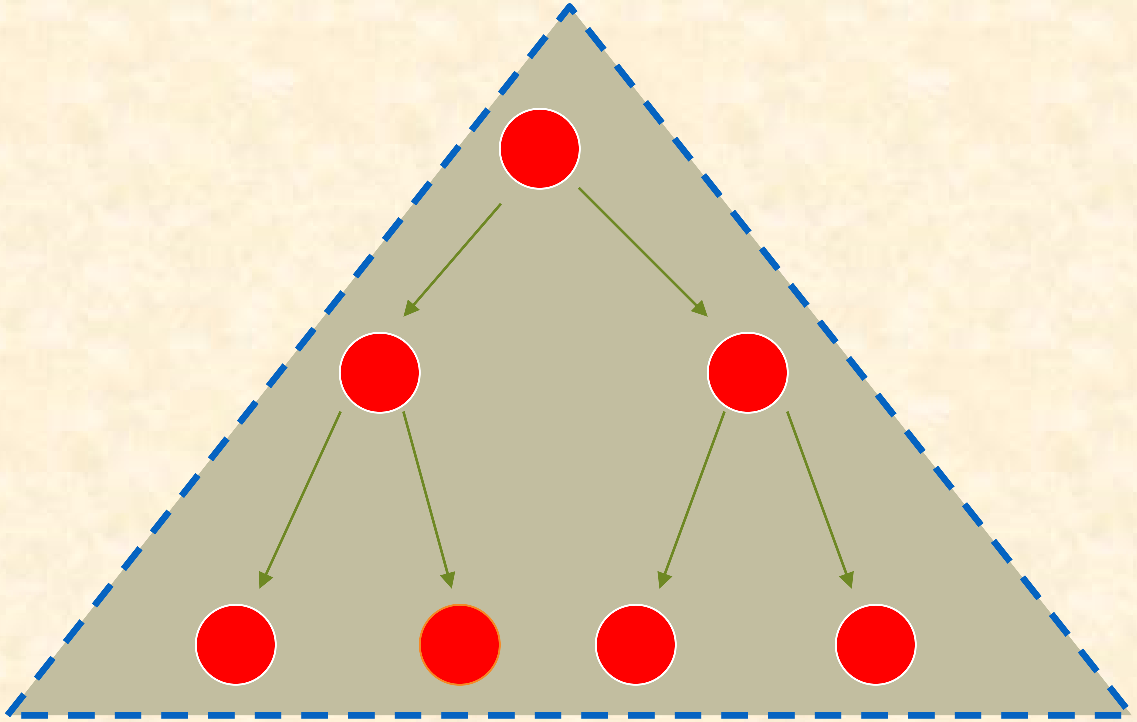
CTL: Computation Tree Logic



EG red

For at least a path, red remains always true”

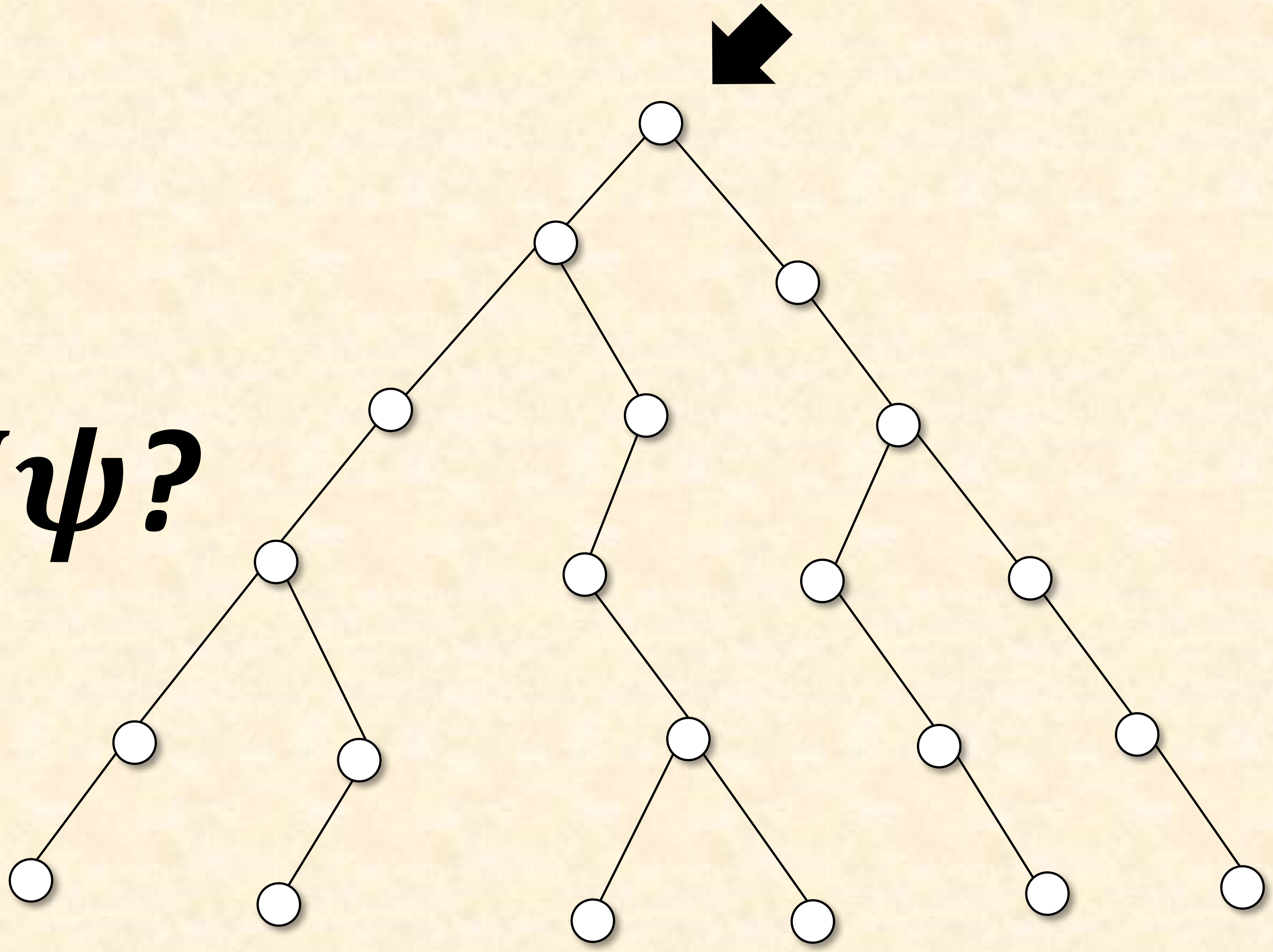
CTL: Computation Tree Logic



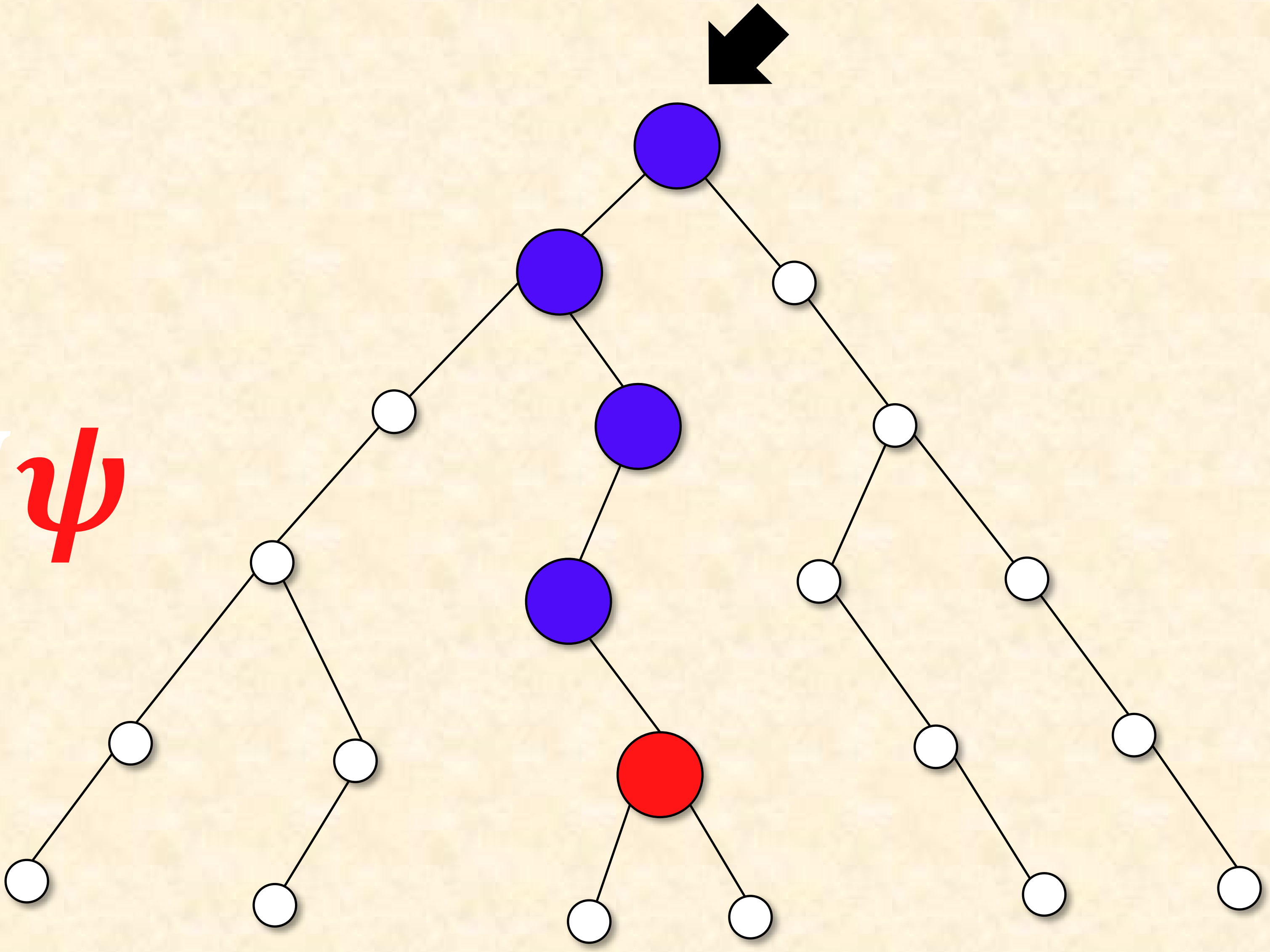
AG red

"on every path, red is always true"

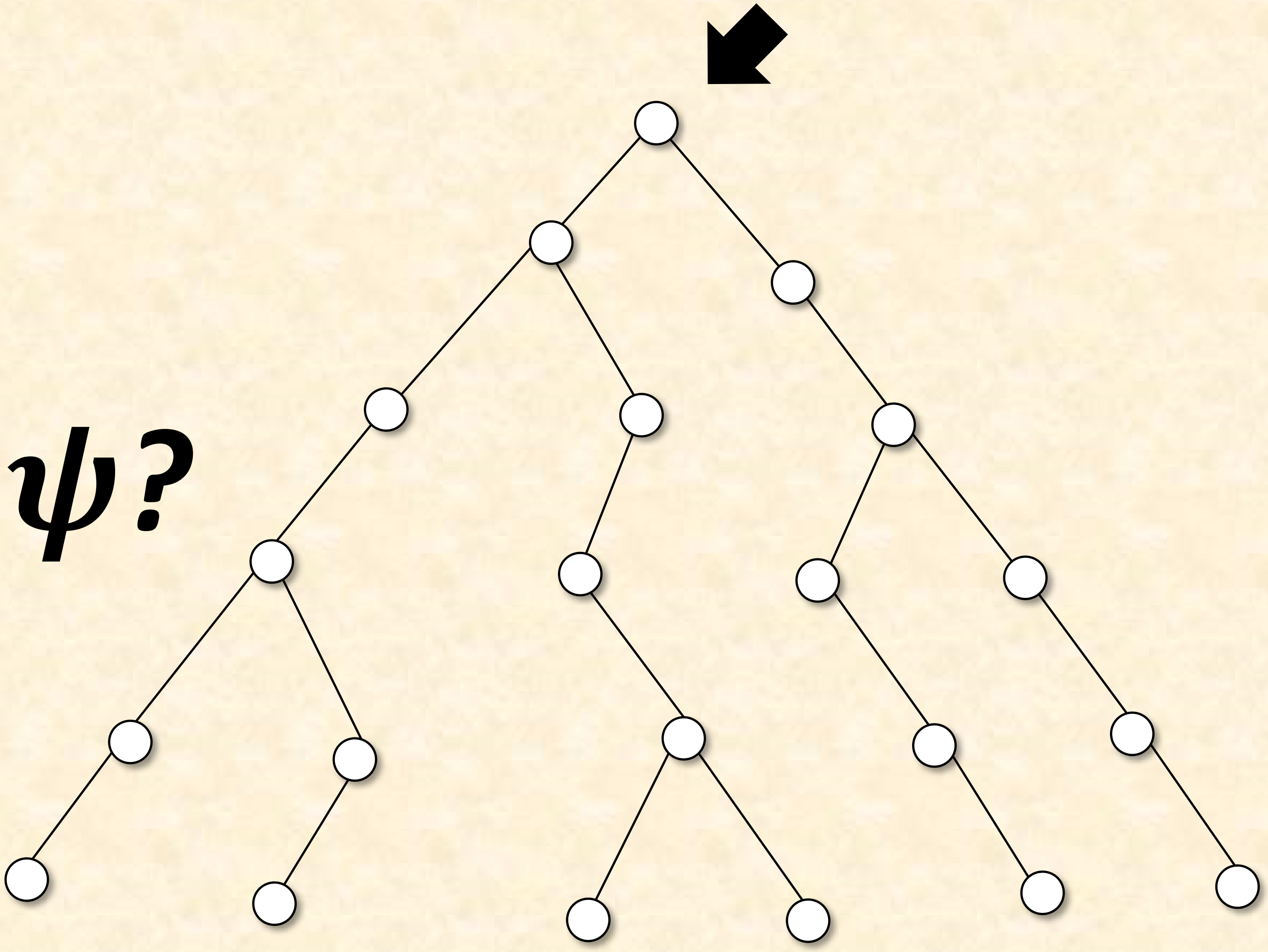
$$\mathcal{K} \vDash E\varphi U\psi?$$



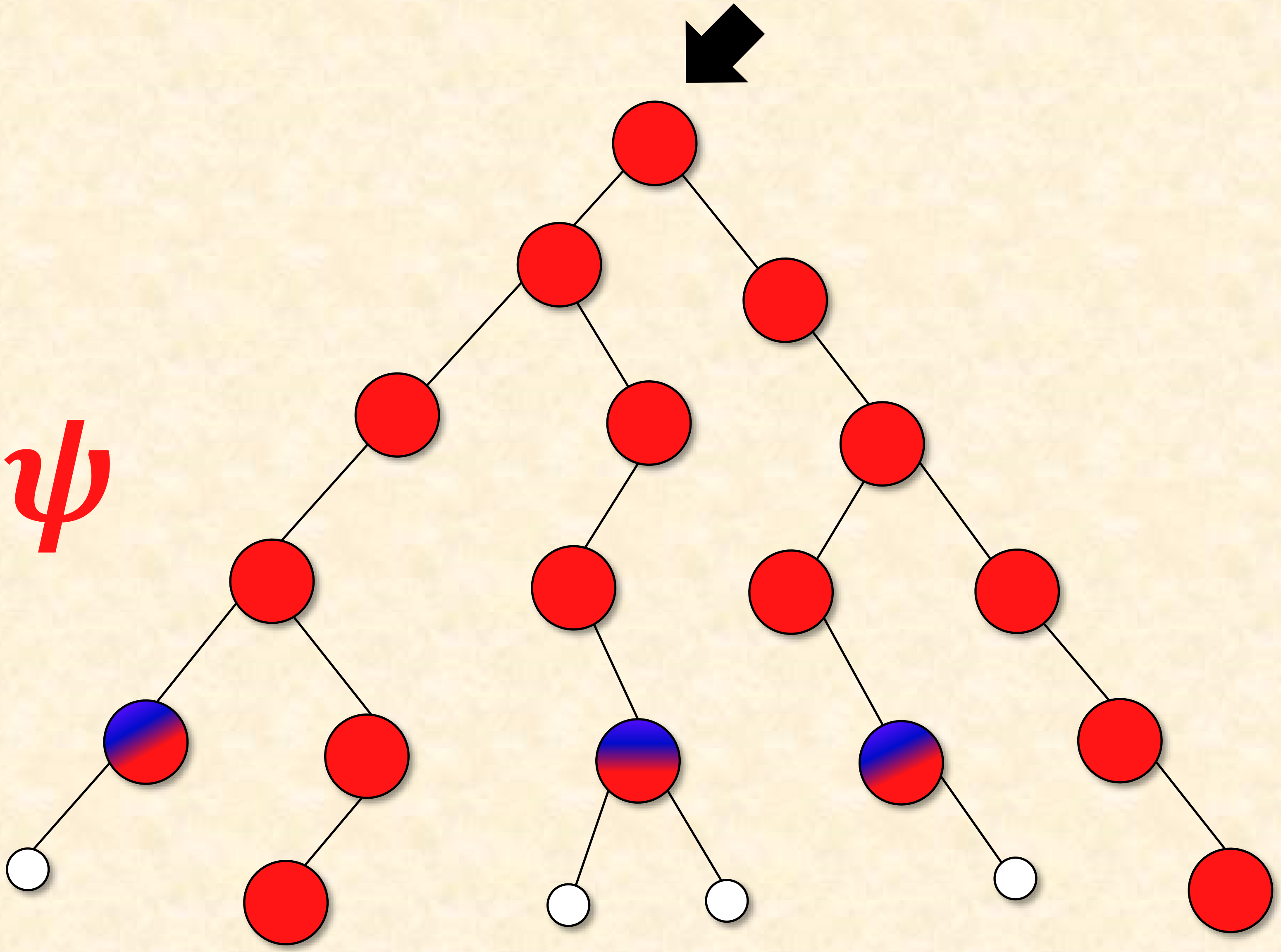
$\mathcal{K} \models E\varphi U\psi$



$$\mathcal{M} \vDash A\varphi R\psi?$$



$\mathcal{M} \vDash A\varphi R\psi$



Part 1.1

- ✓ Introduction to formal verification;
- ✓ Models for closed systems: Kripke Structures;
- ✓ Linear and branching-time temporal logics: LTL, CTL, and CTL*;
- → **An automata-theoretic approach to model checking: word and tree automata**

Decision Problems Using Automata

Model Checking

Decision Problems Using Automata

Model Checking

- Given an automaton A_M for the system model M and an automaton $A_{\neg\phi}$ accepting all models of the complement of a specification ϕ, M is correct with respect to $\boldsymbol{\phi}$ **iff**

$$L(A_M) \cap L(A_{\neg\varphi}) = \emptyset$$

Decision Problems Using Automata

Satisfiability

Decision Problems Using Automata

Satisfiability

- Given a temporal logic specification ϕ, using an automaton A_ϕ accepting all models of ϕ, we have that ϕ is satisfiable **iff**

$$L(A_\phi) \neq \Phi$$

Automata-Theoretic approach

- In order to use an automata-theoretic approach, we need to discuss:

Automata-Theoretic approach

- In order to use an automata-theoretic approach, we need to discuss:
- Which kind of automata
  - Branching mode: deterministic – nondeterministic – universal – alternating.
  - Acceptance mode: Buchi – co-Buchi – parity – Streett – Rabin – Muller
  - Input: words – trees

Automata-Theoretic approach

- In order to use an automata-theoretic approach, we need to discuss:
- Which kind of automata
  - Branching mode: deterministic – nondeterministic – universal – alternating.
  - Acceptance mode: Buchi – co-Buchi – parity – Streett – Rabin – Muller
  - Input: words – trees
- How to implement model and specification translations

Automata-Theoretic approach

- In order to use an automata-theoretic approach, we need to discuss:
- Which kind of automata
  - Branching mode: deterministic – nondeterministic – universal – alternating.
  - Acceptance mode: Buchi – co-Buchi – parity – Streett – Rabin – Muller
  - Input: words – trees
- How to implement model and specification translations
- How to check the (non-)emptiness problem

Büchi Word Automata (NBW) [1/2]

- For LTL model checking, we can use Büchi word automata (**NBW**)
- NBW extend classical finite automata in order **to accept ω-words**
- An NBW is a tuple $A = < Q, \Sigma, \delta, Q_0, F >$
  - Q is the set of states
  - $Q_0 \subseteq Q$ is the set of initial states
  - Σ is the alphabet
  - $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition relation (note, it is **nondeterministic**)
  - $F \subseteq Q$ is an **acceptance condition for infinite words**, defined w.r.t. runs

Büchi Word Automata (NBW) [1/2]

- For LTL model checking, we can use Büchi word automata (**NBW**)
- NBW extend classical finite automata in order **to accept ω-words**
- An NBW is a tuple $A = < Q, \Sigma, \delta, Q_0, F >$
  - Q is the set of states
  - $Q_0 \subseteq Q$ is the set of initial states
  - Σ is the alphabet
  - $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition relation (note, it is **nondeterministic**)
  - $F \subseteq Q$ is an **acceptance condition for infinite words**, defined w.r.t. runs
- A **run ρ** over an ω-word Σ-labeled (when it exists) is a Q-labeled ω-word, build in accordance with δ, whose first state is q_0
- A word is **accepted** if there exists and accepting run (**next slide**)
- The language L of A, denoted L(A), is the set of all words accepted by A

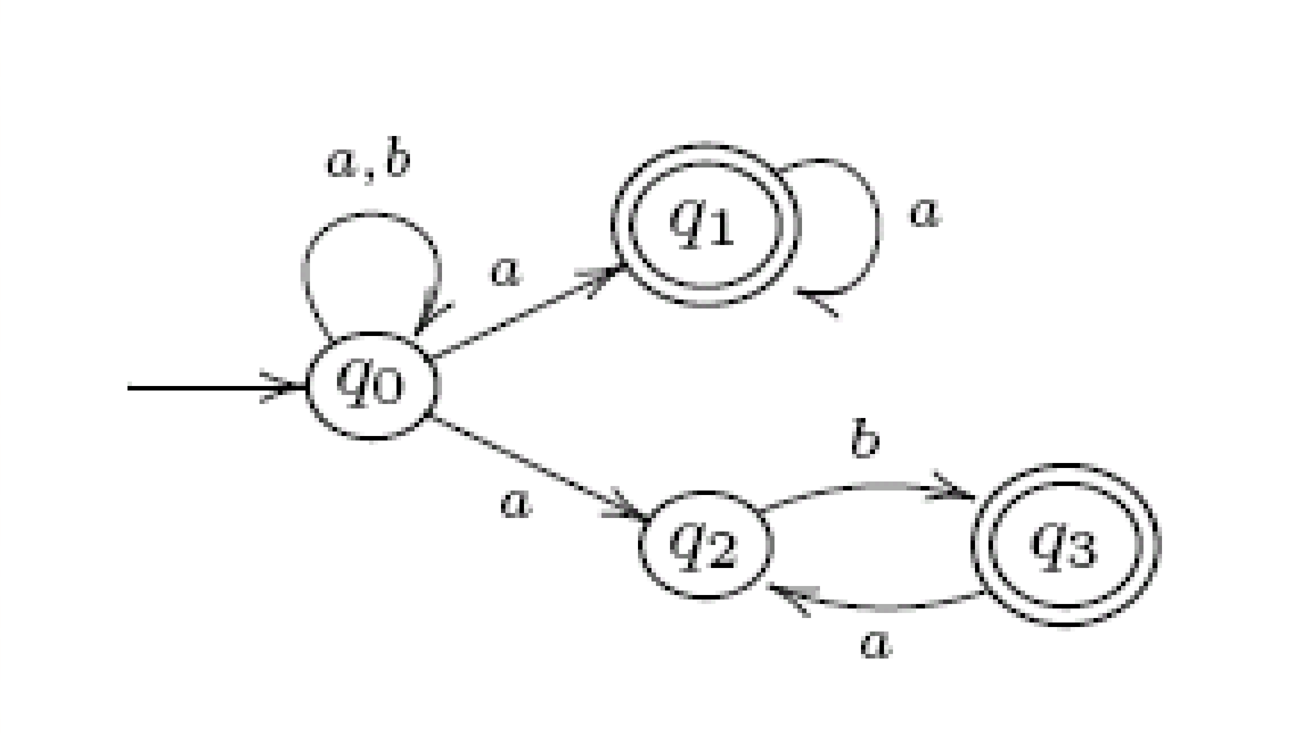
Büchi Word Automata (NBW) [2/2]

- Let $inf(\rho) = \{q \mid q$ appears infinitely often on $\rho\}$,
- A word $\alpha \in \Sigma^*$ is accepted by an NBW A (with $F \subseteq Q$) iff there is a run ρ of A on α s.t.

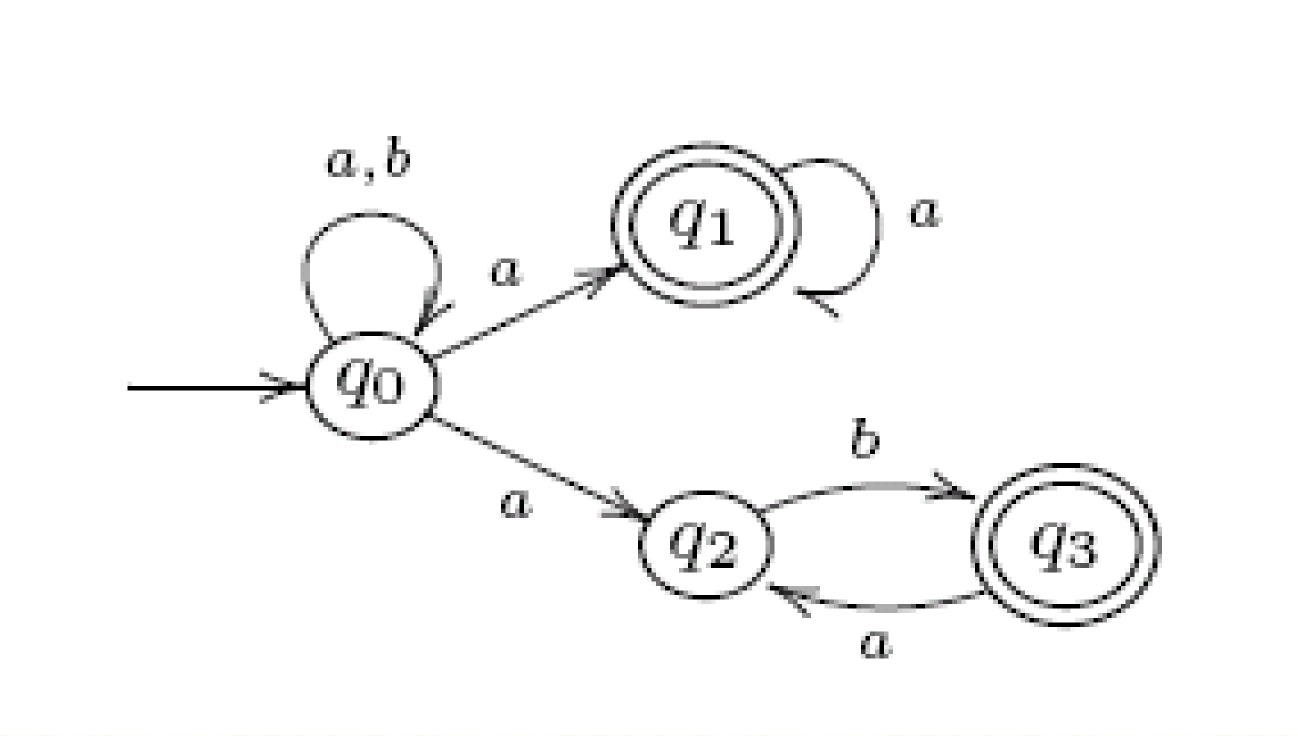
$$Inf(\rho) \cap F \neq \varnothing$$

- In other words, α is accepted by A iff there is a run of A on α visiting a final state $q \in F$ infinitely often
- Such a run is called an accepting run.

Example



Example



L := {$\alpha \in$ {a, b}$^\omega$| α ends with a^ω or with $(ab)^\omega$}

An Automata Approach to LTL Model Checking

- Recall that, given a Model M and an LTL formula φ we check whether $M \models \varphi$ by checking whether:

$$\mathbf{L(A_M) \cap L(A_{\neg\varphi}) = \emptyset}$$

From a Kripke structure to a Buchi automaton

- Given a Kripke structure

$$M= (AP, S, S_0, R, Lab)$$

- …… we can build an equivalent Buchi automaton

$$A_M = < Q, \Sigma, \delta, Q_0, F >$$

- Where:

From a Kripke structure to a Buchi automaton

- Given a Kripke structure

$$M = (AP, S, S_0, R, Lab)$$

- …… we can build an equivalent Buchi automaton

$$A_M = < Q, \Sigma, \delta, Q_0, F >$$

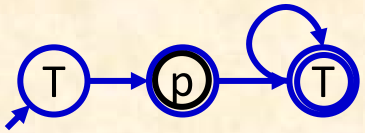
- Where:
  - **$\Sigma = 2^{AP}$**
  - **$Q = S$**: same initial state
  - **(s, a, t) ϵ δ** iff (s,t) **ϵ** R and $a = Lab(s)$
  - **$Q_0 = S_0$** : same initial state
  - **$F = S$** : every state is accepting

From LTL to NBW: Some examples

- Given an LTL formula ϕ we build am NBW A_{ϕ} that accepts all words models of ϕ

From LTL to NBW: Some examples

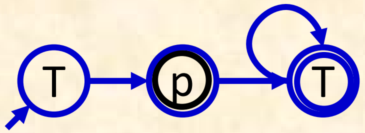
- Given an LTL formula ϕ we build am NBW A_{ϕ} that accepts all words models of ϕ
- Xp



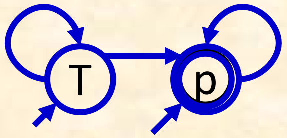
From LTL to NBW: Some examples

- Given an LTL formula ϕ we build am NBW A_ϕ that accepts all words models of ϕ

- Xp

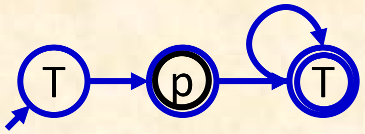


- Fp

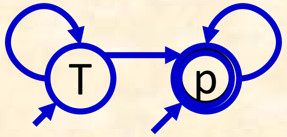


From LTL to NBW: Some examples

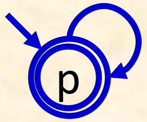
- Given an LTL formula ϕ we build am NBW A_ϕ that accepts all words models of ϕ
- Xp



- Fp

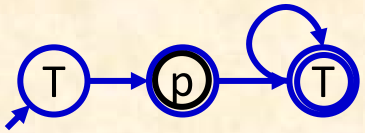


- Gp

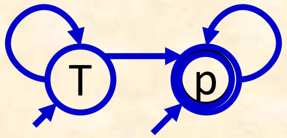


From LTL to NBW: Some examples

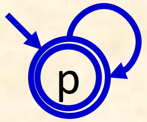
- Given an LTL formula ϕ we build am NBW A_{ϕ} that accepts all words models of ϕ
- Xp



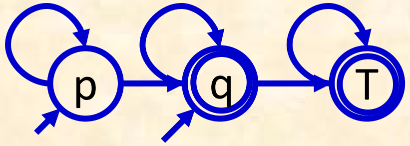
- Fp



- Gp

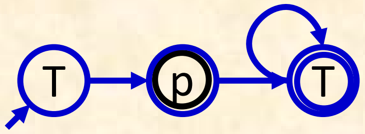


- P U q

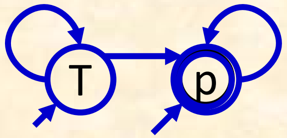


From LTL to NBW: Some examples

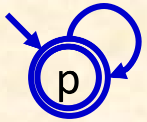
- Given an LTL formula ϕ we build am NBW A_ϕ that accepts all words models of ϕ
- Xp



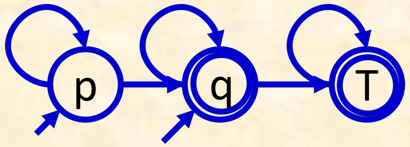
- Fp



- Gp



- P U q



- We skip the formal construction. All you need to know is that the automaton is exponential in the size of the formula (e.g., nesting of temporal operators) [Vardi, Wolper 1986]

An Automata Approach to LTL Model Checking

- Recall that, given a Model M and an LTL formula φ we check whether $M \models \varphi$ by checking whether:

$$L(A_M) \cap L(A_{\neg\varphi}) = \emptyset$$

An Automata Approach to LTL Model Checking

- Recall that, given a Model M and an LTL formula φ we check whether $M \models \varphi$ by checking whether:

$$\mathbf{L(A_M) \cap L(A_{\neg\varphi}) = \emptyset}$$

- The size of $\mathbf{A_M}$ is linear in the size of M

An Automata Approach to LTL Model Checking

- Recall that, given a Model M and an LTL formula φ we check whether $M \models \varphi$ by checking whether:

$$L(A_M) \cap L(A_{\neg\varphi}) = \emptyset$$

- The size of **A_M** is linear in the size of M
- **$A_{\neg\varphi}$** is an NBW and its size is exponential in the size of φ

An Automata Approach to LTL Model Checking

- Recall that, given a Model M and an LTL formula φ we check whether $M \models \varphi$ by checking whether:

$$L(A_M) \cap L(A_{\neg\varphi}) = \emptyset$$

- The size of A_M is linear in the size of M
- $A_{\neg\varphi}$ is an NBW and its size is exponential in the size of φ
- The intersecting language $L(A_M) \cap L(A_{\neg\varphi})$ is the language of an NBW **B** that can be built in PTime.

An Automata Approach to LTL Model Checking

- Recall that, given a Model M and an LTL formula φ we check whether $M \models \varphi$ by checking whether:

$$\mathbf{L(A_M) \cap L(A_{\neg\varphi}) = \emptyset}$$

- The size of **A_M** is linear in the size of M
- **$A_{\neg\varphi}$** is an NBW and its size is exponential in the size of φ
- The intersecting language $L(A_M) \cap L(A_{\neg\varphi})$ is the language of an NBW **B** that can be built in PTime.
- So, **the size of B** is polynomial in the size of M and exponential in the size of φ.

An Automata Approach to LTL Model Checking

- Recall that, given a Model M and an LTL formula φ we check whether $M \models \varphi$ by checking whether:

$$L(A_M) \cap L(A_{\neg\varphi}) = \emptyset$$

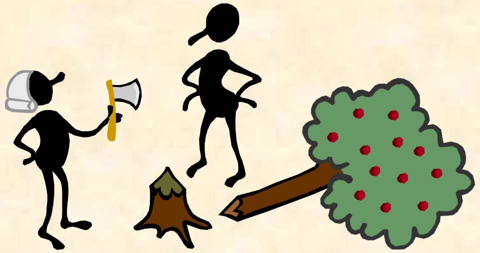
- The size of A_M is linear in the size of M
- $A_{\neg\varphi}$ is an NBW and its size is exponential in the size of φ
- The intersecting language $L(A_M) \cap L(A_{\neg\varphi})$ is the language of an NBW **B** that can be built in PTime.
- So, **the size of B** is polynomial in the size of M and exponential in the size of φ.
- The nonemptiness of B can be checked in LogSpace (look for a **lasso** with double reachability).

An Automata Approach to LTL Model Checking

- Recall that, given a Model M and an LTL formula φ we check whether $M \models \varphi$ by checking whether:

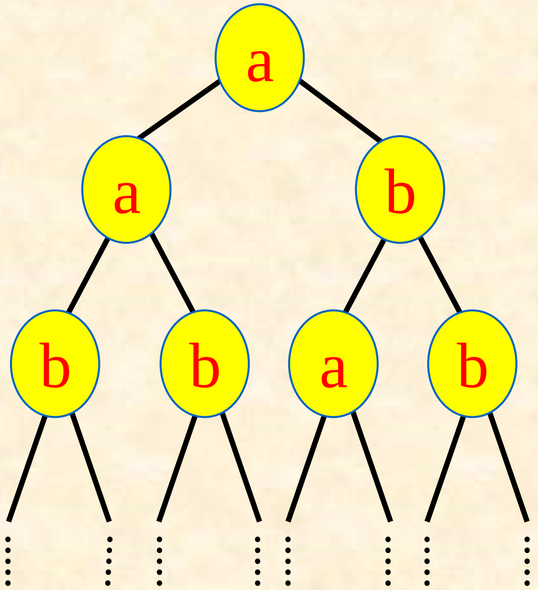
$$L(A_M) \cap L(A_{\neg\varphi}) = \emptyset$$

- The size of **A_M** is linear in the size of M
- **$A_{\neg\varphi}$** is an NBW and its size is exponential in the size of φ
- The intersecting language $L(A_M) \cap L(A_{\neg\varphi})$ is the language of an NBW **B** that can be built in PTime.
- So, **the size of B** is polynomial in the size of M and exponential in the size of φ.
- The nonemptiness of B can be checked in LogSpace (look for a **lasso** with double reachability).
- Finally, we get that model checking question is **PSPACE-complete** and only **PTime** in the size of M



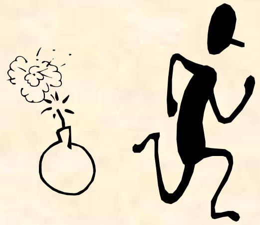
Büchi Tree Automata (NBT)

- For CTL model checking, we can use Büchi tree automata (**NBT**)
- An infinite (binary) tree is **$t : \{0,1\}^* \rightarrow \Sigma$**
- A **path** is an **infinite** sequence of nodes starting at the root
- An **NBT** is a tuple **A = < Q, Σ, δ, Q_0, F >**
  - **δ** : $Q \times \Sigma \rightarrow 2^{Q \times Q}$ is a tree transition relation
  - **F** is an acceptance condition for infinite trees
  - **Acceptance** is defined with respect to runs…. (**next slide**)
- **Note:** we can extend A to deal with any branching degree by means of a **degree** parameter



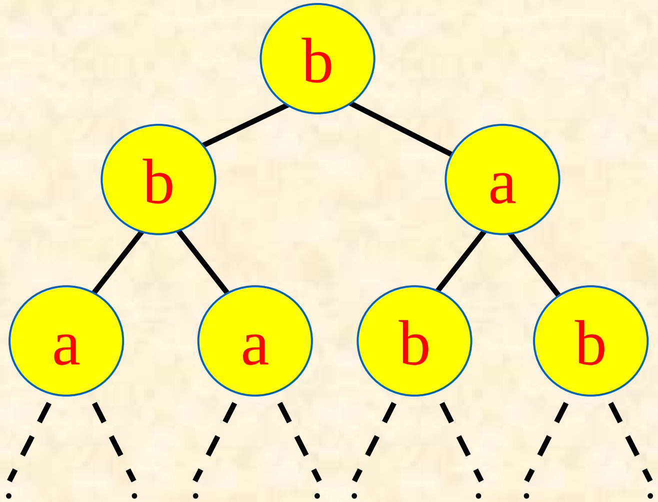
a (Σ-labeled) tree t

Runs

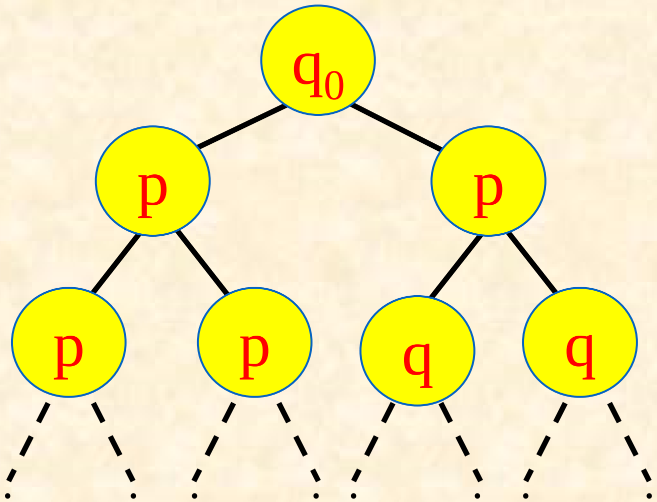


- A **run** r : {0,1}* → Q is built in accordance with δ and r(ε) ∈ Q_0. Thus, runs are Q-labeled trees.

- Let (q,q) ∈ δ(p,a) and q_0 initial state

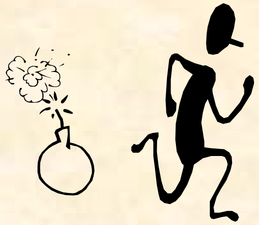


a tree t



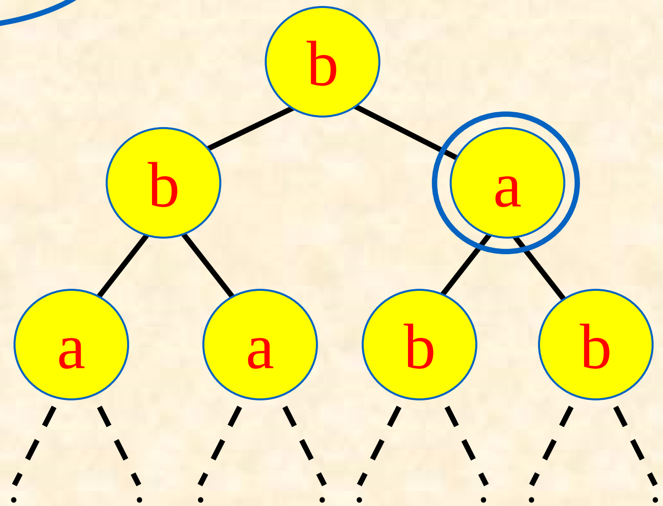
the corresponding run r

Runs

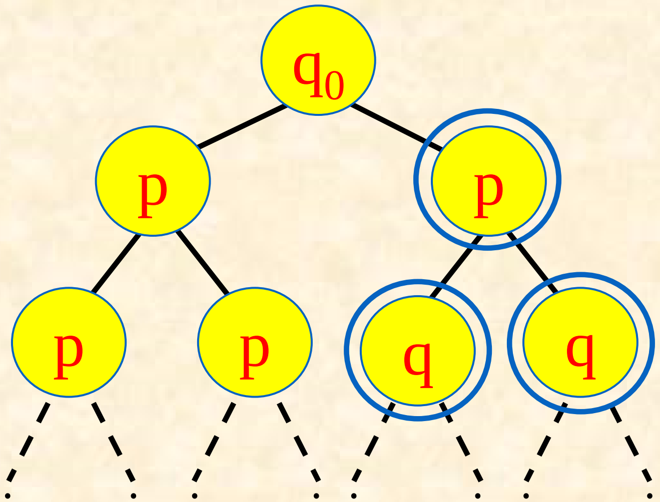


- A run r : $\{0,1\}^* \rightarrow Q$ is built in accordance with δ and $r(\varepsilon) \in Q_0$. Thus, runs are Q-labeled trees.

- Let $(q,q) \in \delta(p,a)$ and q_0 initial state

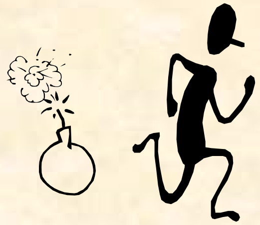


a tree t



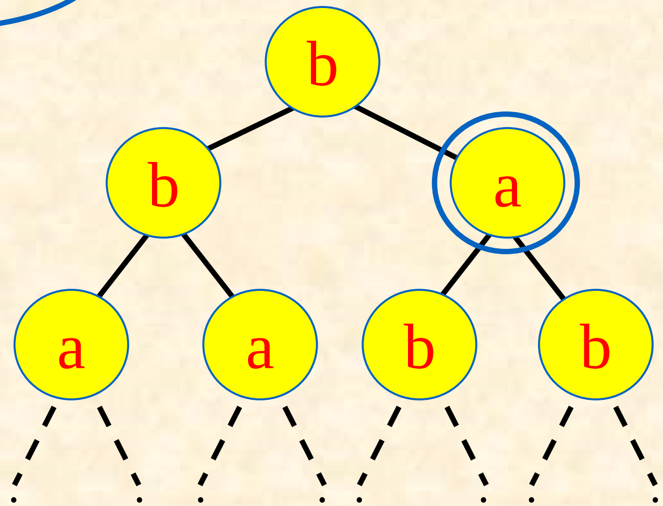
the corresponding run r

Runs

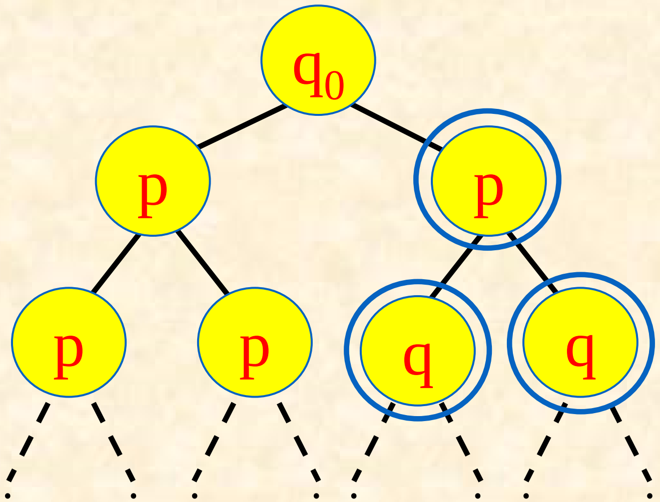


- A run r : $\{0,1\}^* \rightarrow Q$ is built in accordance with δ and $r(\varepsilon) \in Q_0$. Thus, runs are Q-labeled trees.

- Let $(q,q) \in \delta(p,a)$ and q_0 initial state



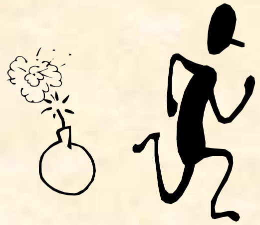
a tree t



the corresponding run r

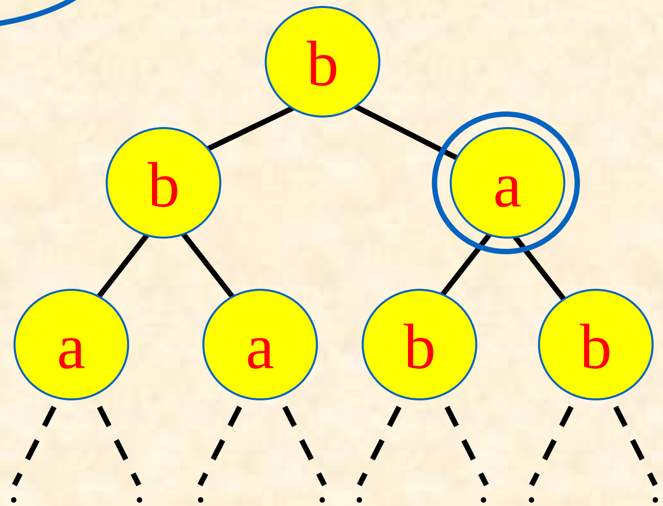
- **Büchi condition ($F \subseteq Q$):**

Runs

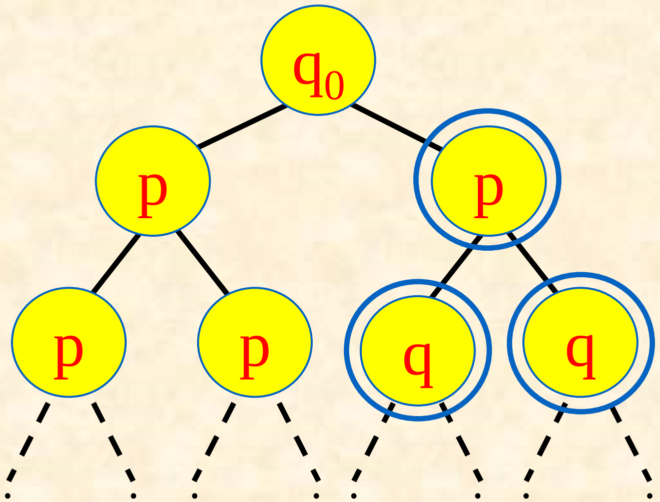


- A **run** r : {0,1}* → Q is built in accordance with δ and r(ε) ∈ Q_0. Thus, runs are Q-labeled trees.

- Let (q,q) ∈ δ(p,a) and q_0 initial state



a tree t



the corresponding run r

- **Büchi condition (F ⊆ Q):**
  - A run **r** is accepting for a **Nonderministic Buchi tree automaton (NBT)** if for every path π

$$\mathrm{Inf}(r|\pi) \cap F \neq \emptyset$$

An Automata Approach to CTL/CTL* Model Checking

An Automata Approach to CTL/CTL* Model Checking

- We can repeat the same argument for NBW and LTL model checking

An Automata Approach to CTL/CTL* Model Checking

- We can repeat the same argument for NBW and LTL model checking

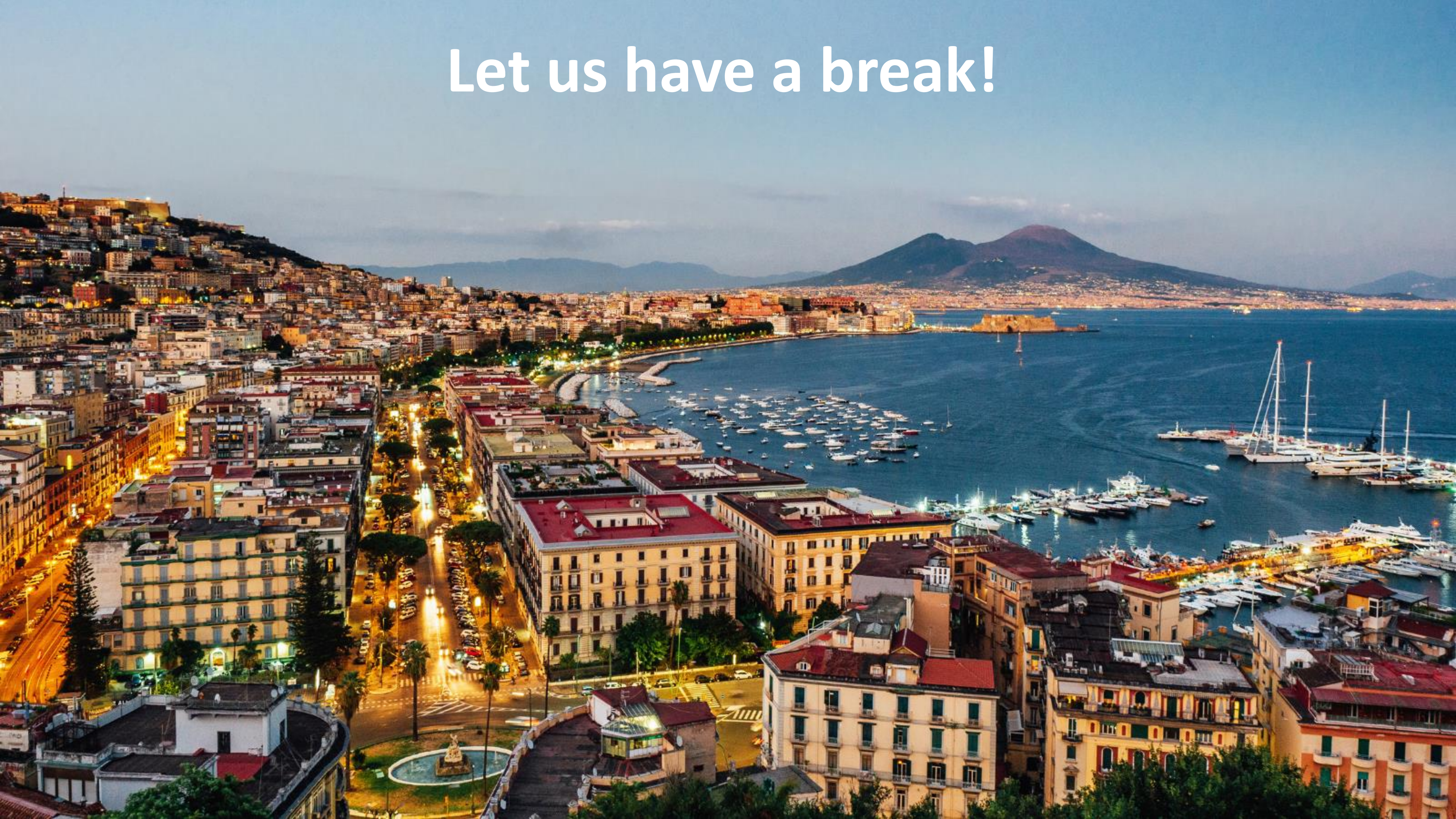
- An efficient upper bound can be obtained via Alternating Buchi tree automata
  - CTL Model checking is PTIME-complete

An Automata Approach to CTL/CTL* Model Checking

- We can repeat the same argument for NBW and LTL model checking

- An efficient upper bound can be obtained via Alternating Buchi tree automata
  - CTL Model checking is PTIME-complete

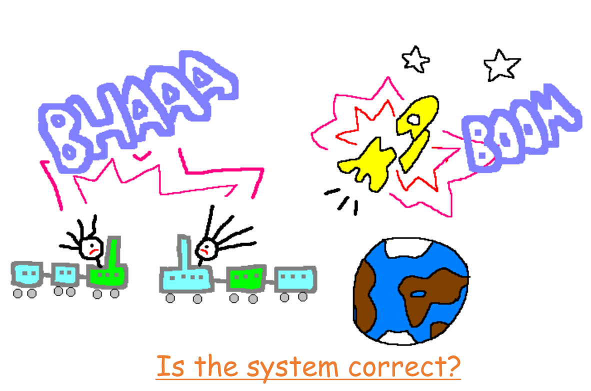
- For CTL* we need a more complex acceptance condition, such as Parity
  - CTL* Model checking is also PSPACE-complete


Let us have a break!

Part 1.2

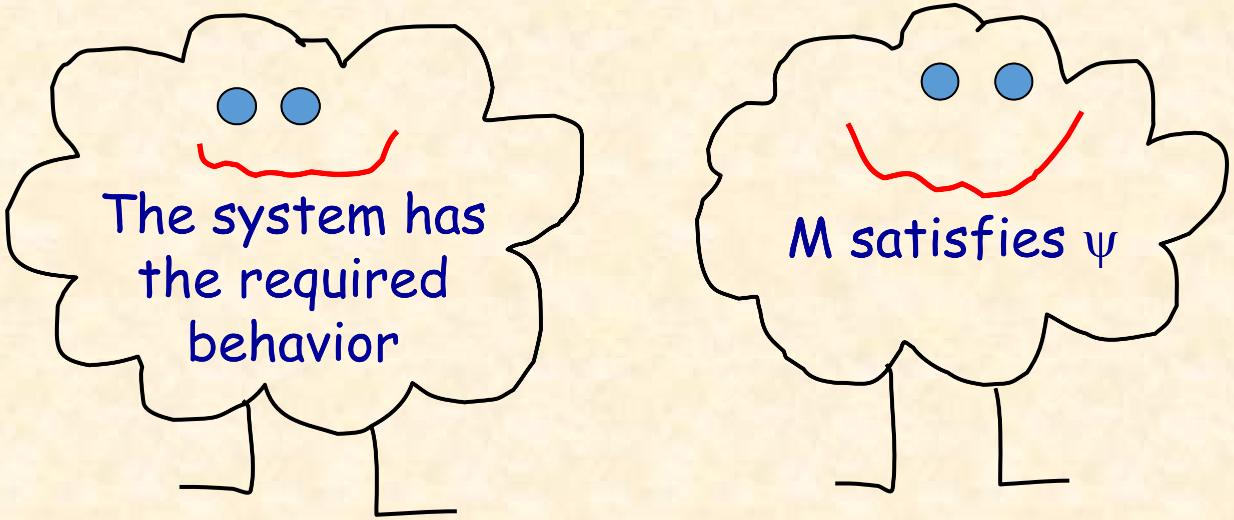
From one player to two players

Model Checking



- Let S be a finite-state system and P its desired behavior

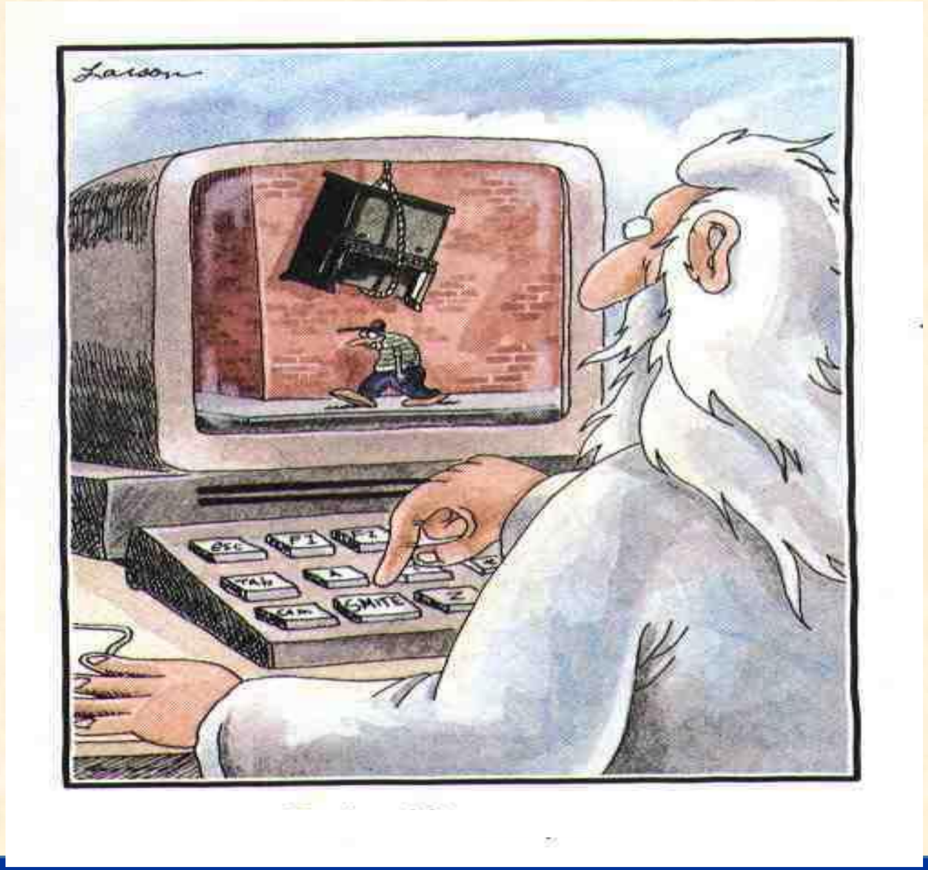
  - S → labelled state-transition graph M
  - P → a temporal logic formula ψ



Picture credits to Orna Kupferman

Classes of Models

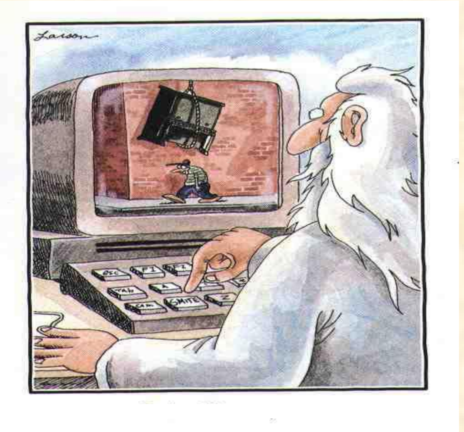
- Closed Systems
  - Behavior is fully characterized by system state
- Open Systems
  - Behavior depends on the interaction with the environment



Classes of Models

- Closed Systems
  - Behavior is fully characterized by system state
- Open Systems
  - Behavior depends on the interaction with the environment

It must be "reactive"

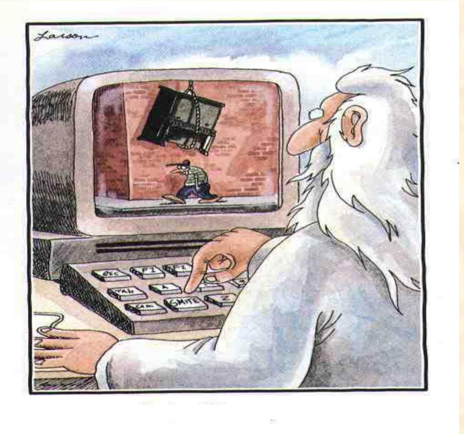


- Open System Model: ~~Labelled State-Transition Graph~~

Classes of Models

- ❑ Closed Systems
  - ➢ Behavior is fully characterized by system state
- ❑ Open Systems
  - ➢ Behavior depends on the interaction with the environment

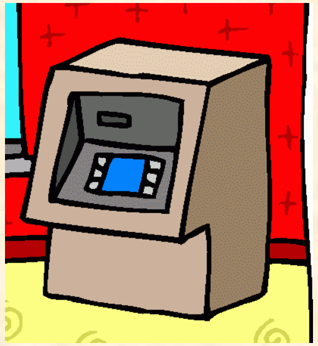
It must be "reactive"



- ➢ Open System Model: ~~Labelled State-Transition Graph~~
- ➢ A solution for Open Finite-State Systems: Module Checking [Kupferman, Vardi, Wolper 2001]

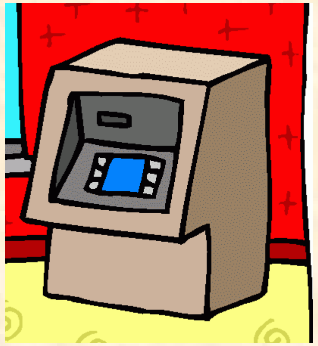
Model checking a closed system

- Consider an ATM machine that
  1. Displays a welcome screen
  2. Makes an internal nondeterministic choice
  3. **Withdraws money** or **shows an advertisement (Ad)**



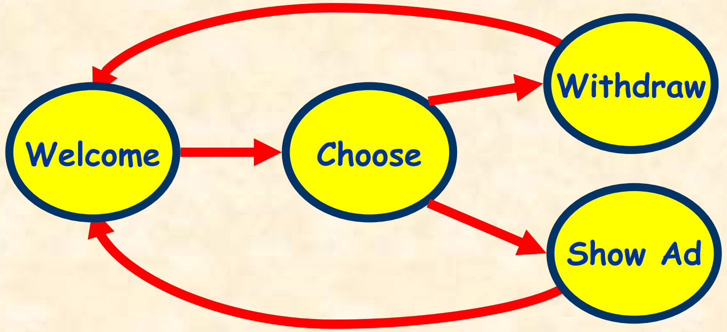
Model checking a closed system

- Consider an ATM machine that
  1. Displays a welcome screen
  2. Makes an internal nondeterministic choice
  3. **Withdraws money** or **shows an advertisement (Ad)**



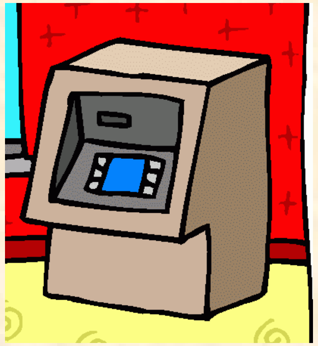
- The machine is a closed system !
- M is a labeled-state transition graph modeling the machine

M:



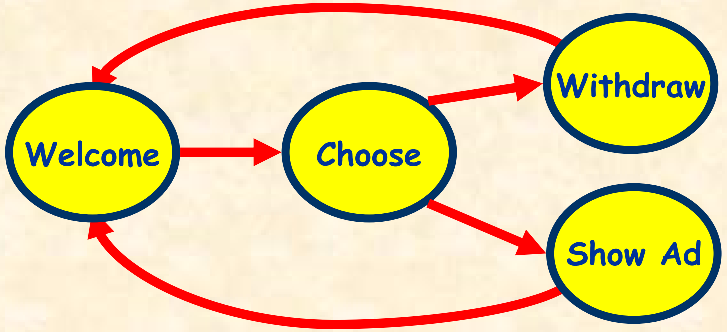
Model checking a closed system

- Consider an ATM machine that
  1. Displays a welcome screen
  2. Makes an internal nondeterministic choice
  3. **Withdraws money** or **shows an advertisement (Ad)**



- The machine is a closed system !
- M is a labeled-state transition graph modeling the machine

M:

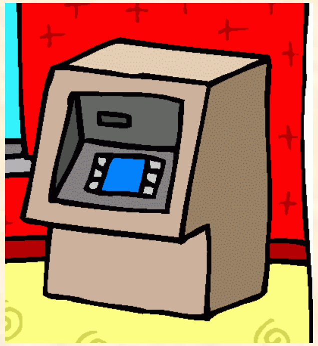


- A desired behavior:

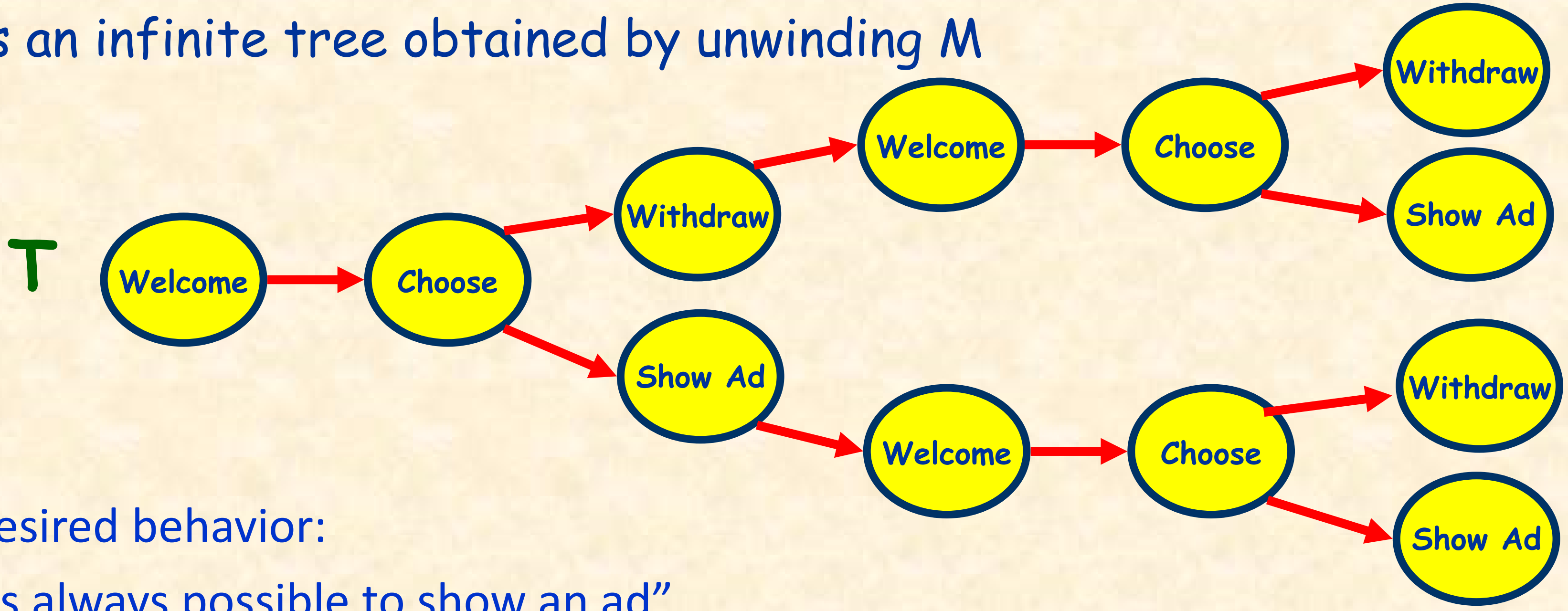
  "It is always possible to show an ad"

$\varphi = \forall G \exists F$ Show Ad

Model checking a closed system



- Consider an ATM machine that
  1. Displays a welcome screen
  2. Makes an internal nondeterministic choice
  3. **Withdraws money** or **shows an advertisement (Ad)**
- The machine is a closed system !
- M is a labeled-state transition graph modeling the machine
- T is an infinite tree obtained by unwinding M



- A desired behavior:

  "It is always possible to show an ad"

$\varphi = \forall G \exists F$ Show Ad ⟹ $M \models \varphi$ iff $T \models \varphi$

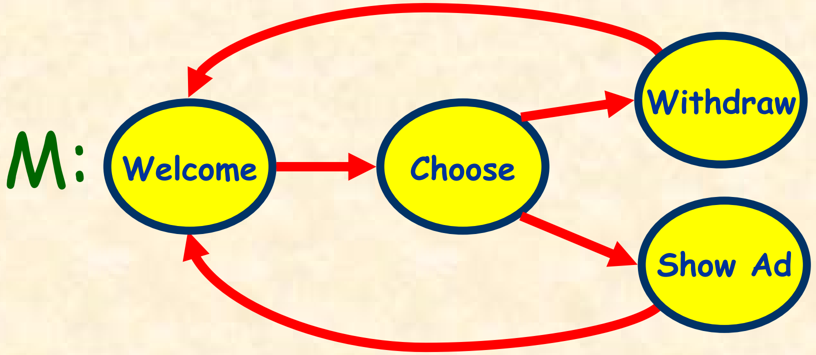
Model checking an open system

- Consider the ATM machine as an open system:
  1. Displays a welcome screen
  2. Lets the environment choose to view an Ad or withdraw money
  3. Performs the requested operation and restarts from 1

Model checking an open system

- Consider the ATM machine as an open system:
  1. Displays a welcome screen
  2. Lets the environment choose to view an Ad or withdraw money
  3. Performs the requested operation and restarts from 1

Open system



- The ATM can always eventually show an Ad iff

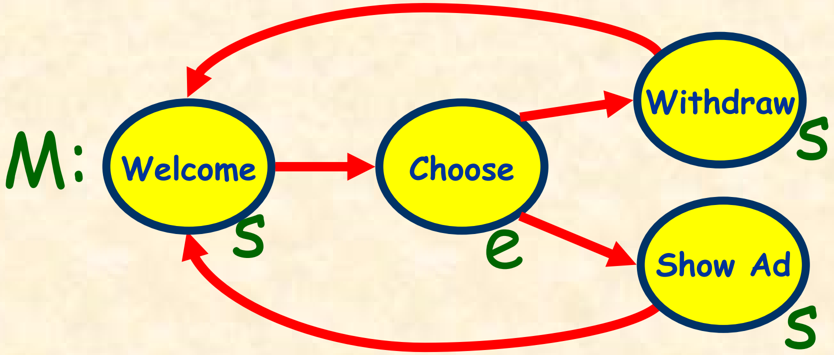
~~$T \models \forall G\, \exists F$ Show Ad~~

It may be impossible to show an ad!

Model checking an open system

- Consider the ATM machine as an open system:
  1. Displays a welcome screen
  2. Lets the environment choose to view an Ad or withdraw money
  3. Performs the requested operation and restarts from 1

Open system

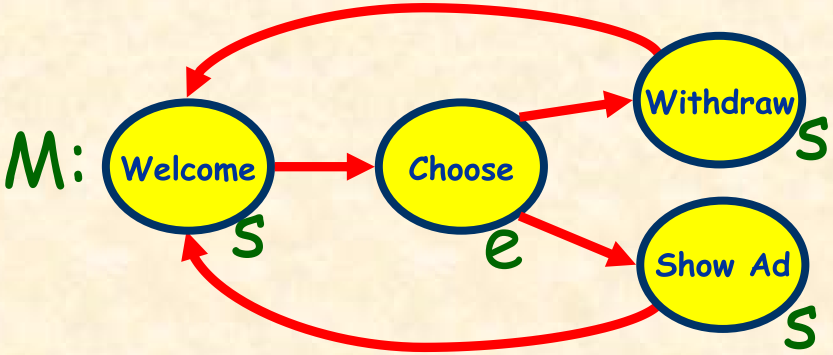


- To model the ATM we need a **Module**: a labeled transition graph with a partition into system and environment nodes

Model checking an open system

- Consider the ATM machine as an open system:
  1. Displays a welcome screen
  2. Lets the environment choose to view an Ad or withdraw money
  3. Performs the requested operation and restarts from 1

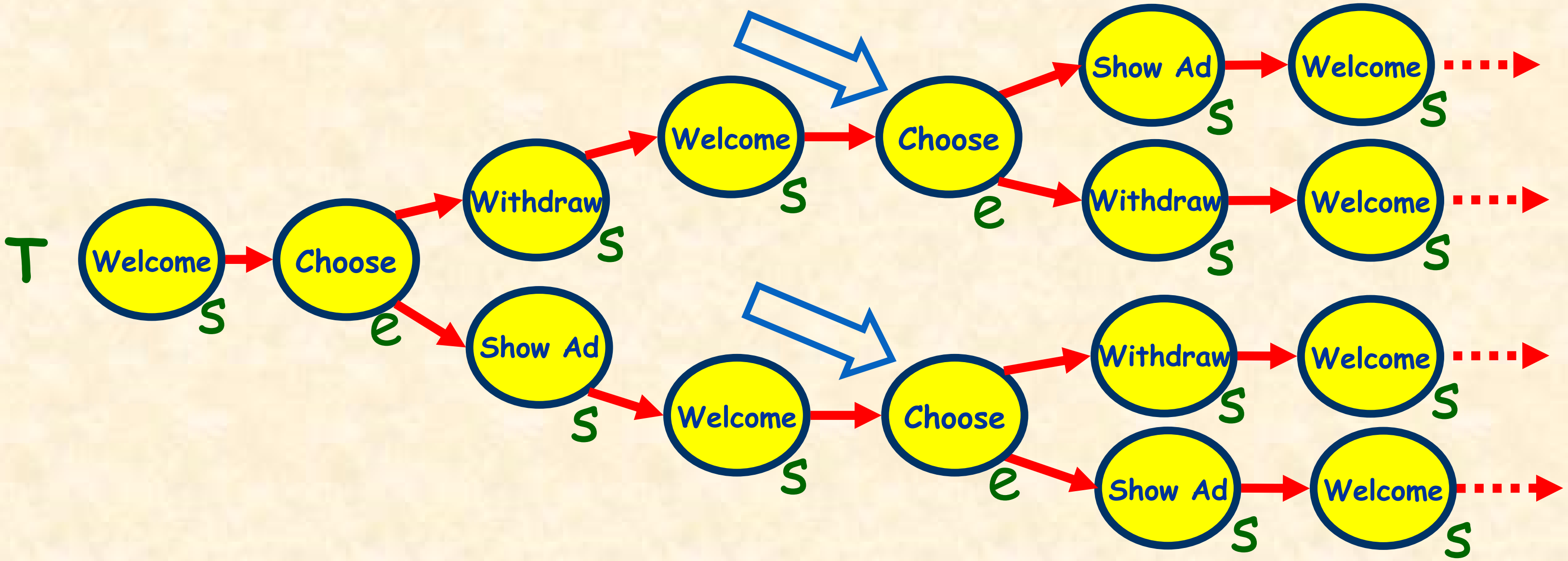
Open system



- To model the ATM we need a **Module**: a labeled transition graph with a partition into system and environment nodes
- Let T be the unwinding of M.
- Let **Exec(M)** be the set of all trees obtained by pruning in T sub-trees rooted in successors of environment nodes (but one).

Model checking an open system

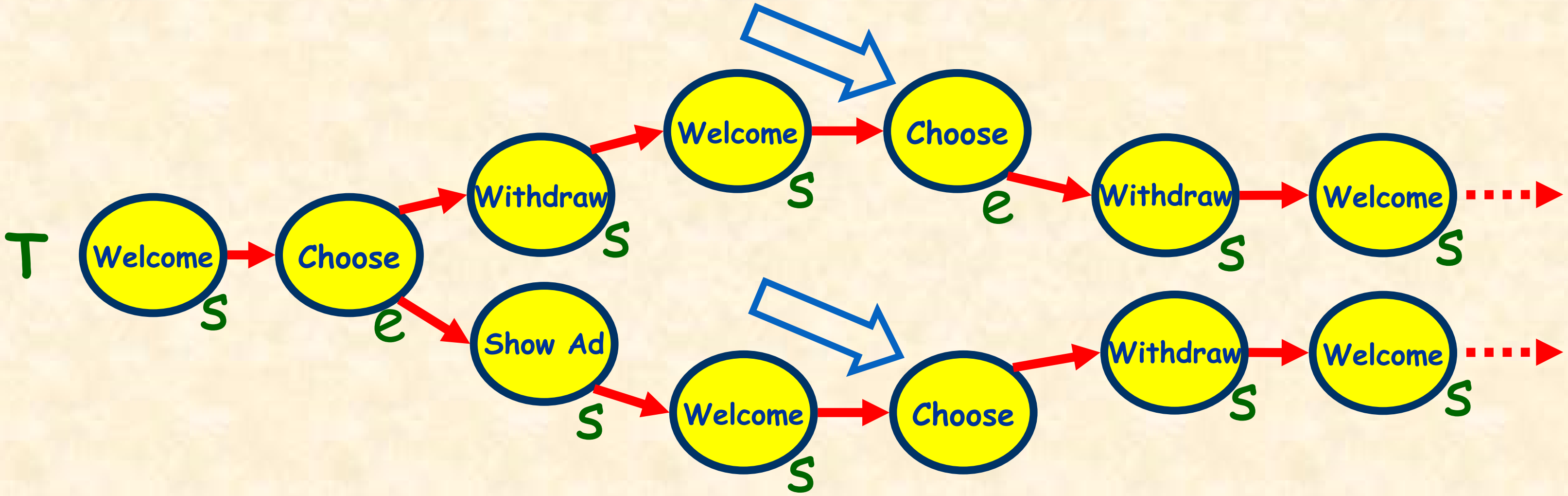
- Consider the ATM machine as an open system:



- To model the ATM we need a **Module**: a labeled transition graph with a partition into system and environment nodes
- Let T be the unwinding of M.
- Let **Exec(M)** be the set of all trees obtained by pruning in T sub-trees rooted in successors of environment nodes (but one).

Model checking an open system

- Consider the ATM machine as an open system:

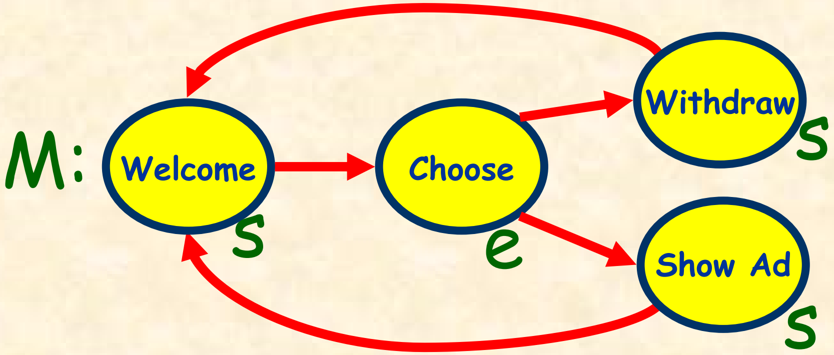


- To model the ATM we need a **Module**: a labeled transition graph with a partition into system and environment nodes
- Let T be the unwinding of M.
- Let **Exec(M)** be the set of all trees obtained by pruning in T sub-trees rooted in successors of environment nodes (but one).

Model checking an open system

- Consider the ATM machine as an open system:
    1. Displays a welcome screen
    2. Lets the environment choose to view an Ad or withdraw money
    3. Performs the requested operation and restarts from 1

Open system



- To model the ATM we need a **Module**: a labeled transition graph with a partition into system and environment nodes
- Let T be the unwinding of M.
- Let **Exec(M)** be the set of all trees obtained by pruning in T sub-trees rooted in successors of environment nodes (but one).
- **M (reactively) satisfies φ iff φ holds in all trees of Exec(M).**

Module checking

$\mathbf{M \models_r \varphi}$

General Observations

- In open systems, the environment can modify internal variables.
- The executions of the system depend on this modification
- The system must be correct no matter how the environment behaves.
- Each possible environment choice induces a different tree in Exec(M)*
- All such trees must satisfy the specification

* In the MAS framework, Exec(M) can be seen as a nondeterministic outcome

Modules: Formal Definition

- A module is a Kripke structure with a partitioning of the states in system states and environment states

$$M = (AP, W_{Sys}, W_{Env}, s_0, R, Lab)$$

- AP, s_0, R, and Lab are as in Kripke structures
- **W_{Sys} and W_{Env}** are a partitioning of the set of states S
  - $W_{Sys} \cap W_{Env} = \emptyset$
  - $W_{Sys} \cup W_{Env} = S$

Exec(M): Formal Definition

- Let T_M be the S-labeled tree unwinding of M (it is labeled with the states of M)
- A tree t is in Exec(M) if a subtree of T_M build as follow
  - The root of t as in T_M (i.e., it is labeled with s_0)
  - For each node x in t, corresponding to a node $w_s \in W_{Sys}$, the children of x are all successors of w_s in M
  - For each node x in t, corresponding to a node $w_e \in W_{Env}$, children of x are a nonempty subset of successors of w_e in M
- The Size of Exec(M) can be infinite!

The Module Checking Problem

- Given a module M and a CTL formula ϕ, we say that M reactively satisfies ϕ, denoted:

$$\mathbf{M} \models_r \phi$$

if all trees in exec(M) satisfy ϕ

The Module Checking Problem

- Given a module M and a CTL formula ϕ, we say that M reactively satisfies ϕ, denoted:

$$M \models_r \phi$$

  if all trees in exec(M) satisfy ϕ

- Note that
  - $M \models_r \phi$ implies $M \models \phi$, while the converse may not be true

Solving Module Checking

- CTL Module Checking is EXPTime-Complete[KVW'01]

Solving Module Checking

- CTL Module Checking is EXPTime-Complete[KVW'01]
- Lower Bound: Reduction from CTL satisfiability (intuition: SAT game simulation)

Solving Module Checking

- CTL Module Checking is EXPTime-Complete[KVW'01]
- Lower Bound: Reduction from CTL satisfiability (intuition: SAT game simulation)
- Upper Bound: automata-theoretic approach via BTA
  - Let M be a module and ϕ be a CTL specification
  - In PTime, we buid a BTA $A_{Exec(M)}$ that accepts all trees in exec(M)
  - In EXPTime, we buid a BTA $A_{\neg\phi}$ that accepts all tree models of $\neg\phi$
  - Then, we check whether $M \models_r \phi$ by checking $L(A_{Exec(M)}) \cap L(A_{\neg\phi}) = \emptyset$

Solving Module Checking

- CTL Module Checking is EXPTime-Complete[KVW'01]
- Lower Bound: Reduction from CTL satisfiability (intuition: SAT game simulation)
- Upper Bound: automata-theoretic approach via BTA
  - Let M be a module and ϕ be a CTL specification
  - In PTime, we buid a BTA $A_{Exec(M)}$ that accepts all trees in exec(M)
  - In EXPTime, we buid a BTA $A_{\neg\phi}$ that accepts all tree models of $\neg\phi$
  - Then, we check whether $M \models_r \phi$ by checking $L(A_{Exec(M)}) \cap L(A_{\neg\phi}) = \emptyset$
- The resulting automaton is a BTA, polynomial in |M| and exponential in | ϕ |

Solving Module Checking

- CTL Module Checking is EXPTime-Complete[KVW'01]
- Lower Bound: Reduction from CTL satisfiability (intuition: SAT game simulation)
- Upper Bound: automata-theoretic approach via BTA
  - Let M be a module and ϕ be a CTL specification
  - In PTime, we buid a BTA $A_{Exec(M)}$ that accepts all trees in exec(M)
  - In EXPTime, we buid a BTA $A_{\neg\phi}$ that accepts all tree models of ¬ϕ
  - Then, we check whether $M \models_r \phi$ by checking $L(A_{Exec(M)}) \cap L(A_{\neg\phi}) = \emptyset$
- The resulting automaton is a BTA, polynomial in |M| and exponential in | ϕ |
- The emptiness of a BTA can be checked in quadratic time, therefore:

CTL Module checking is EXPTIME in |ϕ| and PTime in M

Solving Module Checking

- CTL Module Checking is EXPTime-Complete[KVW'01]
- Lower Bound: Reduction from CTL satisfiability (intuition: SAT game simulation)
- Upper Bound: automata-theoretic approach via BTA
  - Let M be a module and ϕ be a CTL specification
  - In PTime, we buid a BTA $A_{Exec(M)}$ that accepts all trees in exec(M)
  - In EXPTime, we buid a BTA $A_{\neg\phi}$ that accepts all tree models of $\neg\phi$
  - Then, we check whether $M \models_r \phi$ by checking $L(A_{Exec(M)}) \cap L(A_{\neg\phi}) = \emptyset$
- The resulting automaton is a BTA, polynomial in |M| and exponential in | ϕ |
- The emptiness of a BTA can be checked in quadratic time, therefore:

CTL Module checking is EXPTIME in |ϕ| and PTime in M

- The construction of $A_{Exec(M)}$ is very clever and interesting by its own!

CTL* Module Checking

- Given a Module M and a CTL* formula ϕ, we can check whether $M \models_r \phi$ as for CTL

CTL* Module Checking

- Given a Module M and a CTL* formula ϕ, we can check whether $M \models_r \phi$ as for CTL
- However, we need a more powerful automaton, such as Parity.

CTL* Module Checking

- Given a Module M and a CTL* formula ϕ, we can check whether $M \models_r \phi$ as for CTL
- However, we need a more powerful automaton, such as Parity.
- Consequently, checking for the emptiness is more expensive: CTL* module checking is **double-exponential** in the size of the specification and **PTime** in the size of the model.

Complexity results

Class	Model Checking	Model C. w.r.t. system	Module Checking	Module C. w.r.t.system
LTL	PSpace-Complete	nlogspace	PSpace-Complete	nlogspace
CTL	Linear Time [1]	nlogspace[3]	EXPTime-Complete	Ptime Exptime (for i.i.)
CTL*	PSpace-Complete [2]	nlogspace[3]	2EXPTime-Complete	Ptime Exptime (for i.i.)

1. [Clarke, Emerson, Sistla 1986]
2. [Emerson and Lei 1985]
3. [Kupferman, Vardi, Wolper 2000]

[Kupferman,Vardi,Wolper 1996 & 2001]
[Kupferman,Vardi, 1997] (for i.i.)

Main References to Module Checking

- *Kuperman, Vardi, Wolper*. **Module Cheking.** Information and Computation 2001 [Pre. Ver. in CAV 1996]
- *Kuperman, Vardi*. **Revisited Module Checking** (with imperfect information). CAV 1997
- *Bozzelli, Murano, Peron.* **Pushdown Module Checking**. Formal Methods in Sys. Des. 2010 [Pre. Ver. LPAR 2005]
- *Ferrante, Murano, Parente*. **Enriched μ-Calculi Module Checking**. Logical Methods in Computer Science 2008. [Pre. Ver. FOSSACS'07 and LPAR'07]
- *Aminof, Legay, Murano, Serre, Vard*i. **Pushdown Module Checking with Imperfect Information**. Information and Computation 2013 [Pre. Ver. in CONCUR 2007 and IFIP TCS 2008
- *Murano, Parente, Napoli*. **Program Complexity in Hierarchical Module Checking**. LPAR 2008
- *Jamroga and Murano.* **On module checking and strategies**. AAMAS 2014
- *Jamroga and Murano.* **Module Checking of Strategic Ability**. AAMAS 2015
- *Bozzelli, Murano, Peron.* **Module Checking of Pushdown Multi-agent Systems**. KR 2020
- *Jamroga, Mittelmann, Murano, Perelli.* **Playing Quantitative Games Against an Authority: On the Module Checking Problem**. AAMAS 2024