Model Checking

Verifying ω-Regular Properties

[Baier & Katoen, Chapter 4.4]

Joost-Pieter Katoen and Tim Quatmann

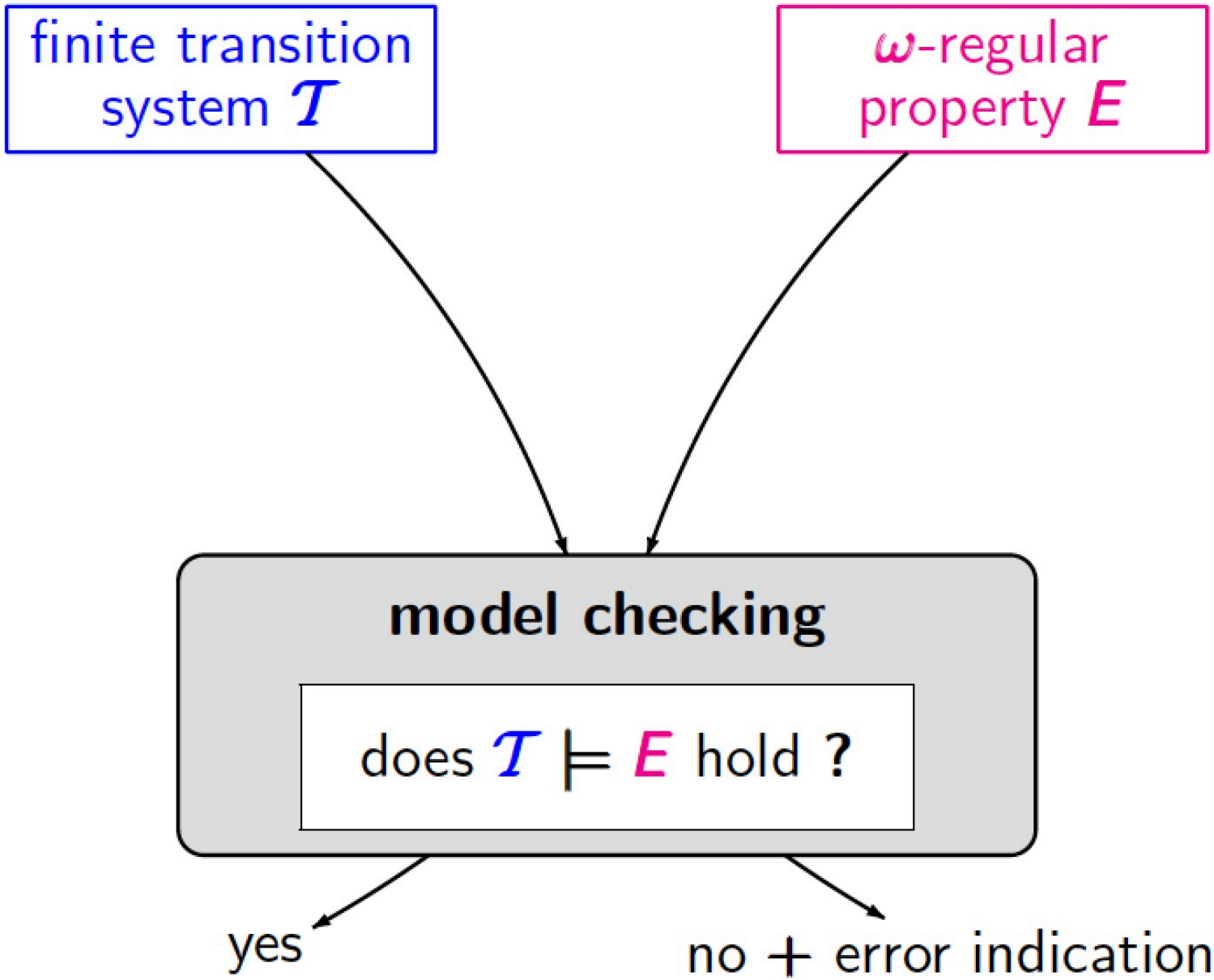
Software Modeling and Verification Group

RWTH Aachen, SoSe 2022

Overview

Overview

Topic


finite transition system $\mathcal{T}$
ω-regular property E
model checking
does $\mathcal{T} \models E$ hold ?
yes
no + error indication

ω-Regular Properties

Definition: ω-regular language

The set $\mathfrak{L}$ of infinite words over the alphabet Σ is ω-regular if $\mathfrak{L} = \mathfrak{L}_\omega(G)$ for some ω-regular expression G over Σ.

Definition: ω-regular properties

LT property E over AP is ω-regular if E is an ω-regular language over 2^{AP}.

This is equivalent to:

LT property E over AP is ω-regular if E is accepted by a non-deterministic Büchi automaton (over the alphabet 2^{AP}).

Nondeterministic Büchi automata

Definition: Nondeterministic Büchi automaton

A nondeterministic Büchi automaton (NBA) $\mathfrak{A} = (Q, \Sigma, \delta, Q_0, F)$ with:

- Q is a finite set of states
- Σ is an alphabet
- $\delta : Q \times \Sigma \to 2^Q$ is a transition function
- $Q_0 \subseteq Q$ a set of initial states
- $F \subseteq Q$ is a set of accept (or: final) states.

This definition is the same as for NFA.

The acceptance condition of NBA is different though.

Language of a Büchi Automaton

- NBA $\mathfrak{A} = (Q, \Sigma, \delta, Q_0, F)$ and infinite word $w = A_1 A_2 \ldots \in \Sigma^\omega$
- A run for w in $\mathfrak{A}$ is an infinite sequence $q_0 q_1 \ldots \in Q^\omega$ such that:
  - $q_0 \in Q_0$ and $q_i \xrightarrow{A_{i+1}} q_{i+1}$ for all $0 \leq i$
- Run $q_0 q_1 \ldots$ is accepting if $q_i \in F$ for infinitely many i

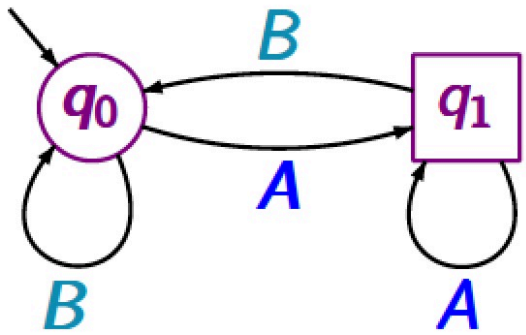
$q_0\ q_1\ q_2\ q_3\ q_4\ q_5 \ldots$ $\in F$ $\in F$ $\in F$ ...

- The accepted language of $\mathfrak{A}$:

$$\mathfrak{L}_\omega(\mathfrak{A}) = \{ w \in \Sigma^\omega \mid \mathfrak{A} \text{ has an accepting run for } w \}$$

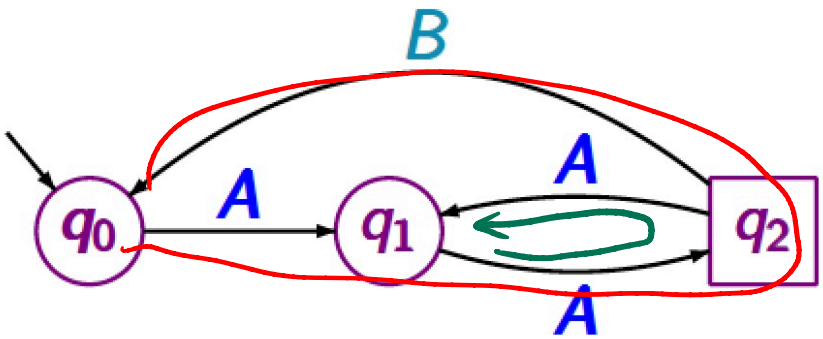
- NBA $\mathfrak{A}$ and $\mathfrak{A}'$ are equivalent if $\mathfrak{L}_\omega(\mathfrak{A}) = \mathfrak{L}_\omega(\mathfrak{A}')$

Examples



accepted language:
set of all infinite words that contain infinitely many ***A***'s

$$(B^*.A)^\omega$$



accepted language:
"every ***B*** is preceded by a positive even number of ***A***'s"

$$((A.A)^+.B)^\omega + ((A.A)^+.B)^*.A^\omega$$

NBA and ω-Regular Languages

Theorem

1. For every NBA $\mathfrak{A}$, the language $\mathfrak{L}_\omega(\mathfrak{A})$ is ω-regular.
2. For every ω-regular language L, there is an NBA $\mathfrak{A}$ with $L = \mathfrak{L}_\omega(\mathfrak{A})$.

Proof.

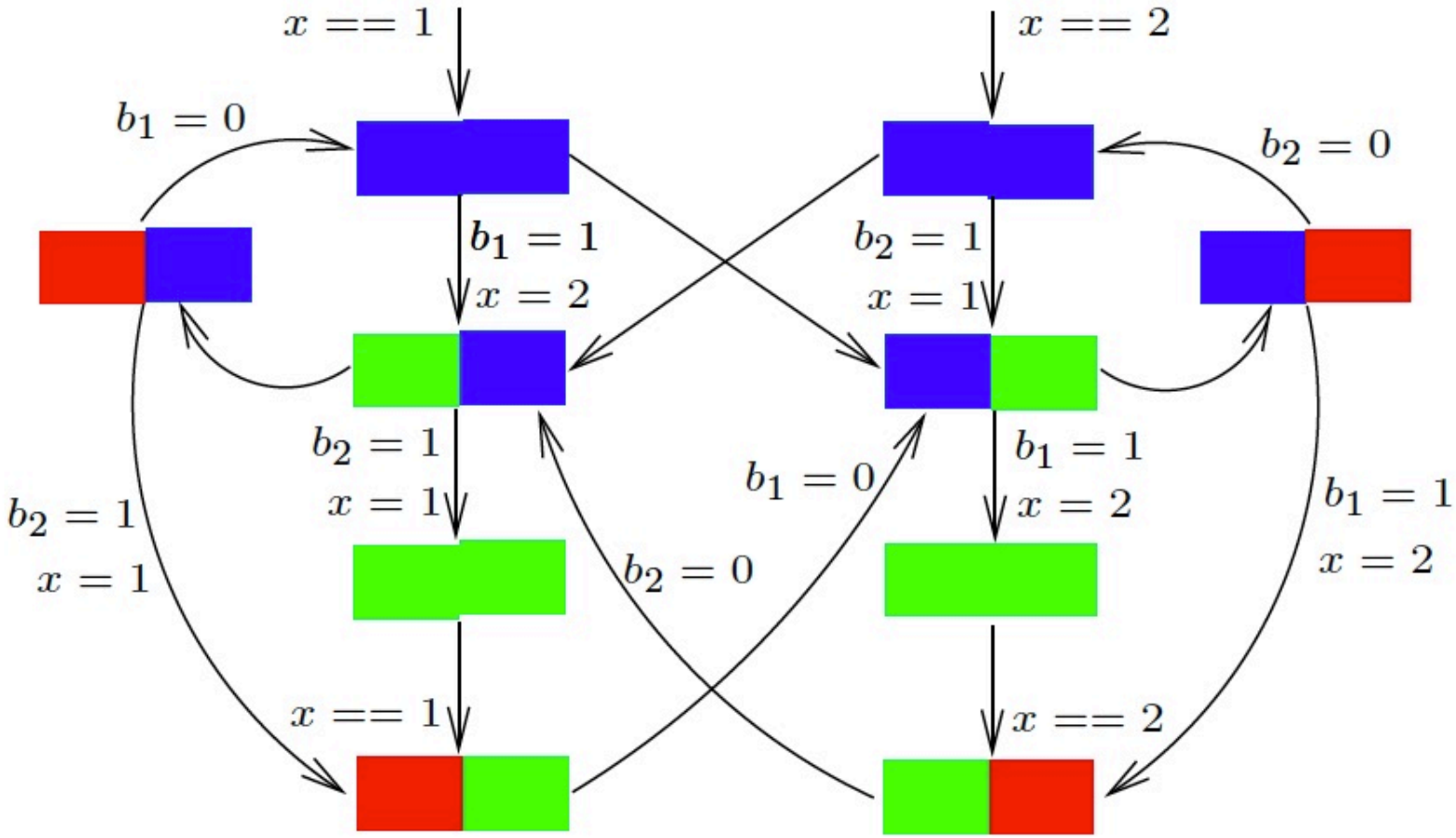
Previous lecture. □

NFA vs. NBA

	NFA	~~DFA~~ NBA
Expressiveness	regular	ω-regular
Closure under ...		
$\cup$	union operator	union operator
$\cap$	product construction	GNBA product
complement	get DFA; $F' := Q \setminus F$	complex procedure ↳ e.g. safra constr.
Determinism	NFA $\equiv$ DFA	DBA $\subsetneq$ NBA
Check $\mathfrak{L}(\mathfrak{A}) = \varnothing$?	DFS / BFS	nested DFS ↳ today
Minimization	unique minimal DFA	—

Overview

Peterson's Transition System



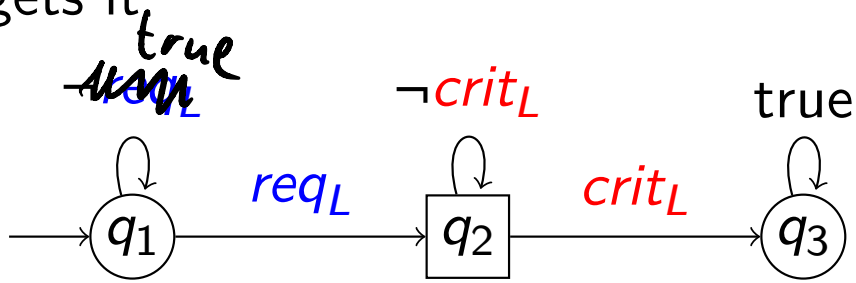
If a thread wants to update the account, does it ever get the opportunity to do so?

"always ($req_L \Rightarrow$ eventually @account$_L$) $\wedge$ always ($req_R \Rightarrow$ eventually @account$_R$)"

Verifying Starvation Freedom

- Starvation freedom = when a thread wants access to account, it eventually gets it

- "Infinite bad prefix" automaton: once a thread wants access to the account, it never gets it



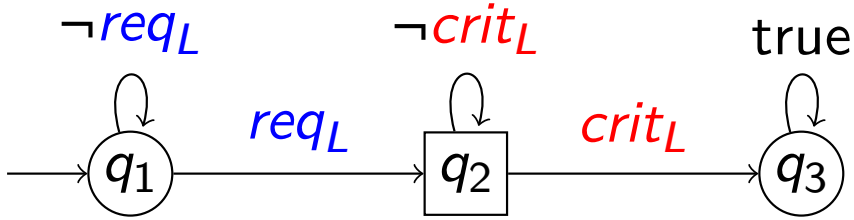
Verifying Starvation Freedom

- Starvation freedom = when a thread wants access to account, it eventually gets it

E_{live}

- "Infinite bad prefix" automaton: once a thread wants access to the account, it never gets it

$\overline{E_{live}}$



- Checking starvation freedom:

$$\underbrace{\mathit{Traces}(\mathit{TS}_{Pet})}_{\text{infinite traces}} \cap \mathfrak{L}_\omega(\overline{E_{live}}) = \varnothing?$$

- Intersection, complementation and emptiness of $\underbrace{\text{Büchi automata}}_{\text{accept infinite words}}$

Basic Idea

ω-regular

$$TS \not\models E \quad \text{if and only if} \quad Traces(TS) \not\subseteq E$$

$$\text{if and only if} \quad Traces(TS) \cap \left(2^{AP}\right)^{\omega} \setminus E \neq \emptyset$$

$\rightsquigarrow$ $\exists$ violating trace

$$\text{if and only if} \quad Traces(TS) \cap \overline{E} \neq \emptyset$$

$\overline{E}$: ω-regular

Basic Idea

$$
\begin{aligned}
TS \not\models E \quad & \text{if and only if} \quad Traces(TS) \not\subseteq E \\
& \text{if and only if} \quad Traces(TS) \cap \left(2^{AP}\right)^{\omega} \setminus E \neq \varnothing \\
& \text{if and only if} \quad Traces(TS) \cap \overline{E} \neq \varnothing \\
& \text{if and only if} \quad Traces(TS) \cap \mathfrak{L}_{\omega}(\mathfrak{A}) \neq \varnothing \\
& \text{if and only if} \quad TS \otimes \mathfrak{A} \not\models \underbrace{\text{“eventually for ever” } \neg F}_{\text{persistence property}}
\end{aligned}
$$

where $\mathfrak{A}$ is an NBA accepting the complement property $\overline{E} = \left(2^{AP}\right)^{\omega} \setminus E$

Persistence Property

Definition: persistence property

A persistence property over AP is an LT property $E_{pers} \subseteq \left(2^{AP}\right)^{\omega}$ of the form "eventually for ever Φ" for some propositional logic formula Φ over AP:

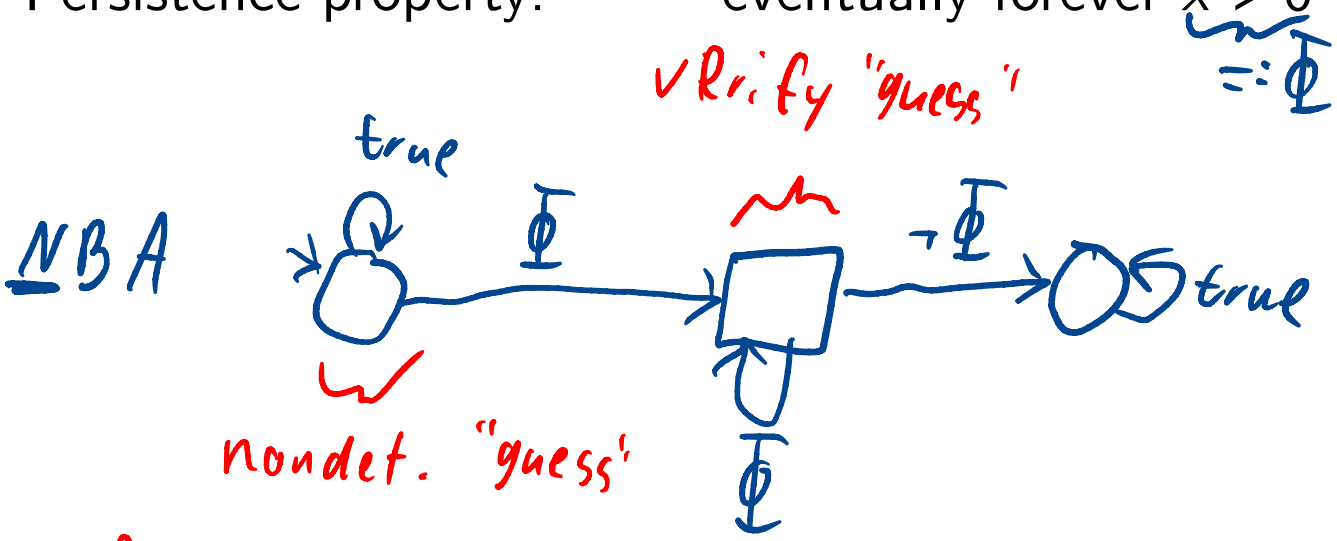
$$E_{pers} = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} \mid \exists i \geq 0.\ \forall j \geq i.\ A_j \vDash \Phi \right\}$$

The formula Φ is called the persistence (or state) condition of E_{pers}.

"Φ is an invariant after a while"

Example

Persistence property: "eventually forever $x > 0$"



from which point on "invariant" Φ holds

there is no DBA for persistence properties

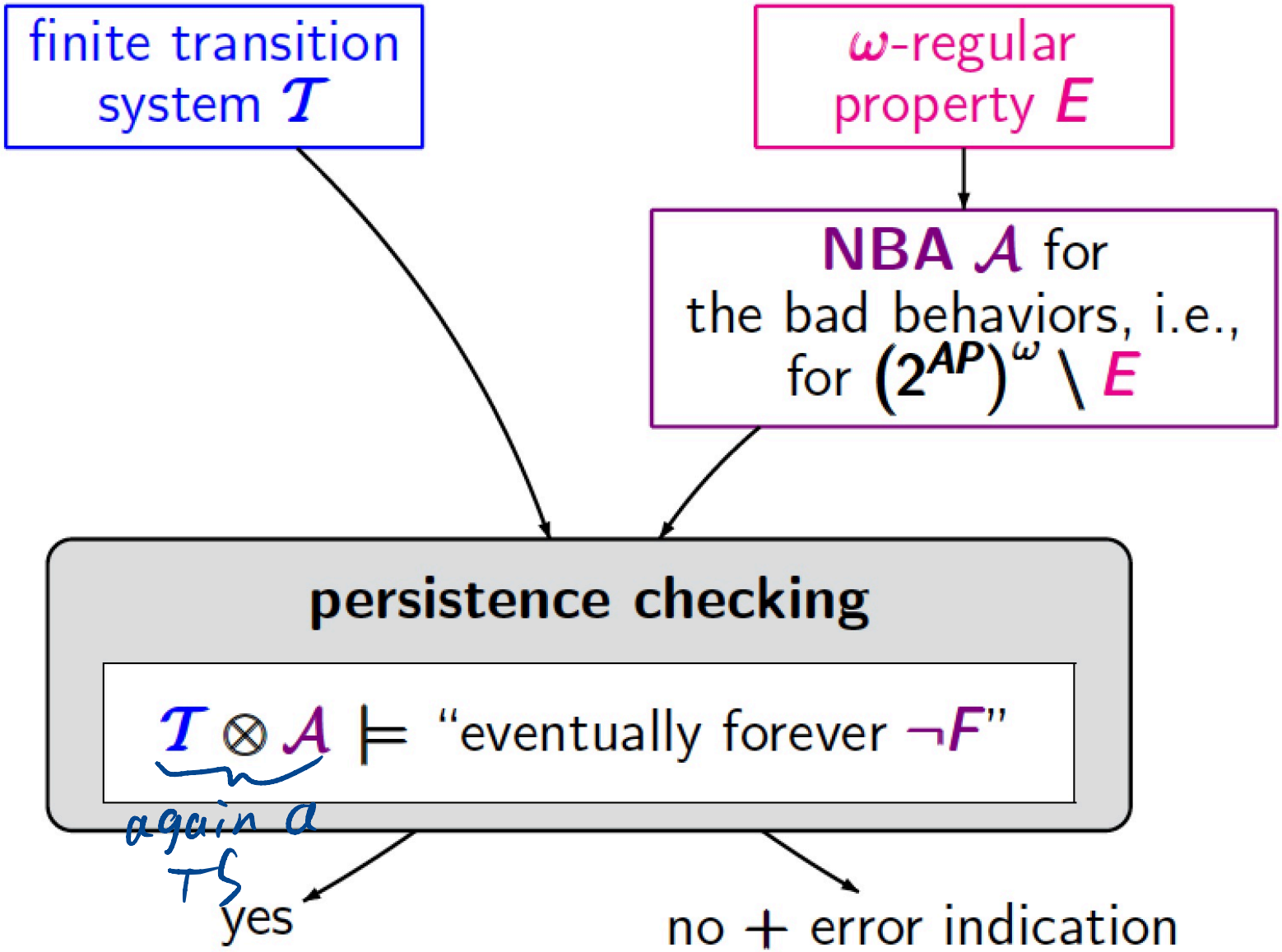
Problem Statement

Let

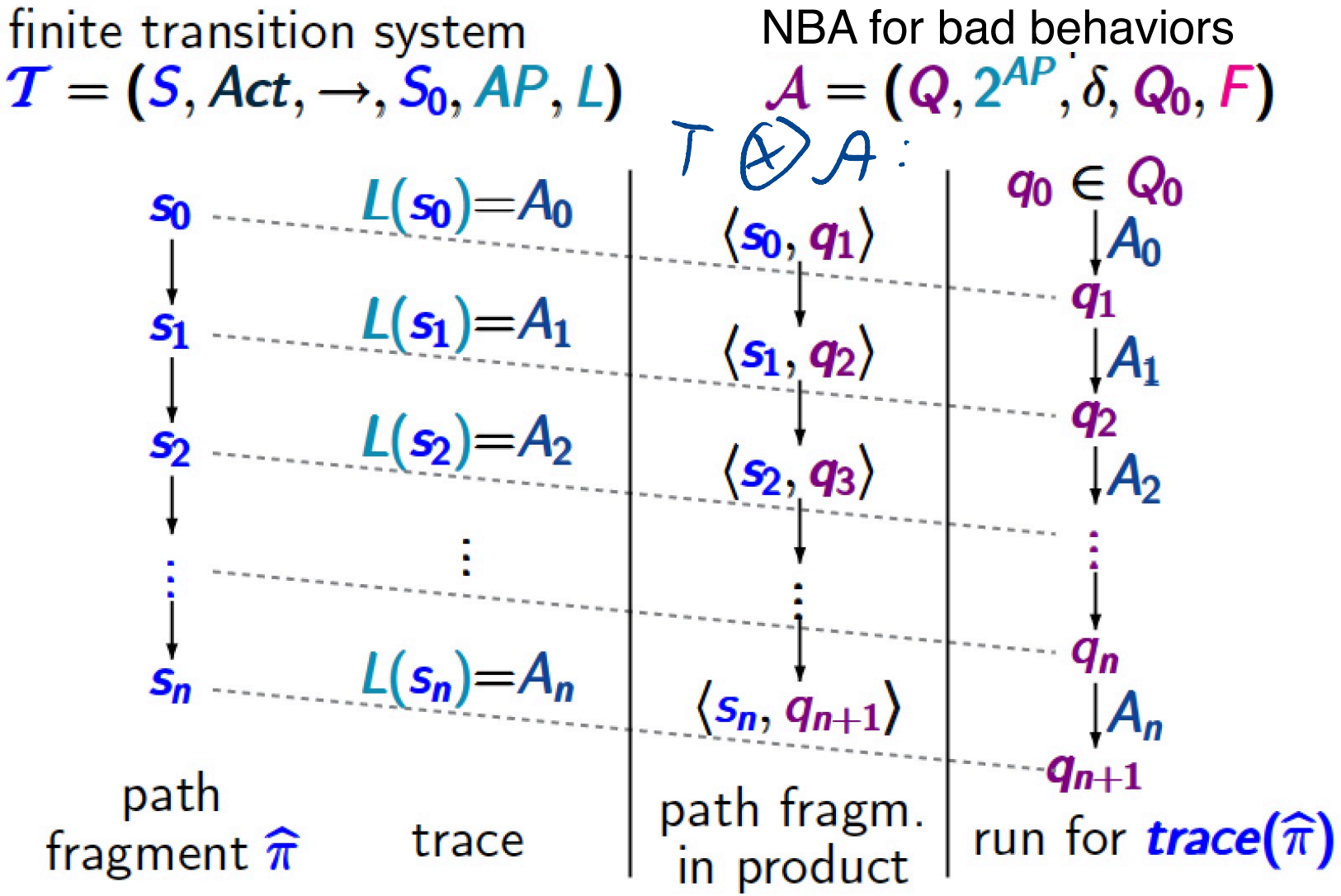
1. E be an ω-regular property over AP
2. $\mathfrak{A}$ be an NBA recognizing the complement of E
3. TS be a finite transition system (over AP) without terminal states

How to establish whether $TS \models E$?

Verifying Omega-Regular Properties



Product: Idea



Synchronous Product

$\forall q \in Q, A \in \Sigma : \delta(q, A) \neq \emptyset$

Definition: synchronous product of TS and NBA

Let transition system $TS = (S, Act, \rightarrow, I, AP, L)$ without terminal states and $\mathfrak{A} = (Q, \Sigma, \delta, Q_0, F)$ a non-blocking NBA with $\Sigma = 2^{AP}$.

The product of TS and $\mathfrak{A}$ is the transition system:

$$TS \otimes \mathfrak{A} \;=\; (S', Act, \rightarrow', I', AP', L') \qquad \text{where}$$

- $S' = S \times Q$, $AP' = Q$ and $L'(\langle s, q \rangle) \;=\; \{ q \}$
- $\rightarrow'$ is the smallest relation defined by:

$$\frac{s \xrightarrow{\alpha} t \;\wedge\; q \xrightarrow{L(t)} p}{\langle s, q \rangle \xrightarrow{\alpha}' \langle t, p \rangle}$$

- $I' = \{ \langle s_0, q \rangle \mid s_0 \in I \wedge \exists q_0 \in Q_0.\ q_0 \xrightarrow{L(s_0)} q \}$.

This coincides with $TS \otimes \hat{\mathfrak{A}}$ for NFA $\hat{\mathfrak{A}}$ (Lecture #4)

Verifying ω-Regular Properties

Theorem

Let TS over AP, E an ω-regular property and NBA $\mathfrak{A}$ with $\mathfrak{L}(\mathfrak{A}) = \overline{E}$.

Then:

$$TS \models E \text{ iff } \mathit{Traces}(TS) \cap \mathfrak{L}_\omega(\mathfrak{A}) = \varnothing \text{ iff } TS \otimes \mathfrak{A} \models \underbrace{\text{eventually forever} \neg F}_{\text{persistence property}}$$

where F stands for $\bigvee_{q \in F} q$.

Show:

$$Traces(TS) \cap \mathfrak{L}_\omega(\mathfrak{A}) \neq \varnothing \text{ iff } TS \otimes \mathfrak{A} \not\models \underbrace{\text{eventually forever} \neg F}_{\text{persistence property}}$$

"$\in$" Assume $TS \otimes \mathcal{A} \not\models \ldots$

Let $\pi = \langle s_0, q_1 \rangle \langle s_1, q_2 \rangle \cdots$ be a path in $TS \otimes \mathcal{A}$ s.t. $tr(\pi) \not\models$ "eventually forever $\neg F$"

Then there are ∞ many indices i with $q_i \in F$.

Let $q_0 \in Q_0$ with $q_0 \xrightarrow{L(s_0)} q_1$ (This state exists since $\langle s_0, q_1 \rangle$ is initial in $TS \otimes \mathcal{A}$)

The sequence $q_0 q_1 q_2 \ldots$ is an accepting run of the NBA $\mathcal{A}$ for the word

$$L(s_0) L(s_1) L(s_2) \ldots = \underbrace{tr(s_0 s_1 s_2 \ldots)}_{\sigma} \in Traces(TS)$$

$\rightsquigarrow \sigma \in \mathfrak{L}_\omega(\mathcal{A})$

$\rightsquigarrow \sigma \in Traces(TS) \cap \mathfrak{L}_\omega(\mathcal{A}) \neq \varnothing$ □

Show:

$$Traces(TS) \cap \mathfrak{L}_\omega(\mathfrak{A}) \neq \varnothing \text{ iff } TS \otimes \mathfrak{A} \not\models \underbrace{\text{eventually forever} \neg F}_{\text{persistence property}}$$

"$\Rightarrow$" Assume $\sigma \in Traces(TS) \cap L_\omega(\mathcal{A})$ exists.

Then some $\pi = s_0 s_1 \ldots$ with $trace(\pi) = \sigma$ exists.

Since $\sigma \in L_\omega(\mathcal{A})$ there is an acc. run $q_0 q_1 q_2 \ldots$ in $\mathcal{A}$, i.e. $q_0 \in Q_0$ and

$$\forall i \geq 0 \quad q_i \xrightarrow{L(s_i)} q_{i+1} \text{ and}$$

$$q_i \in F \text{ for } \infty \text{ many } i \geq 0. \quad (*)$$

Thus, we can combine π and run $q_0 q_1 \ldots$ to a path in $TS \otimes \mathcal{A}$:

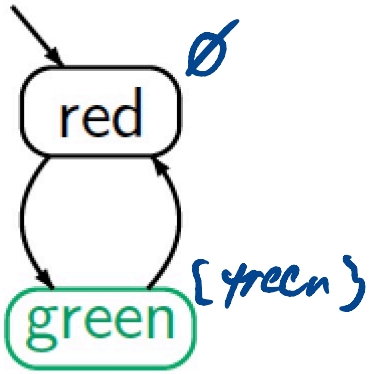
$$\pi' = \langle s_0, q_1 \rangle \langle s_1, q_2 \rangle \ldots$$

Since $(*)$, we have $\pi' \not\models$ "eventually forever $\neg F$"

$\square$

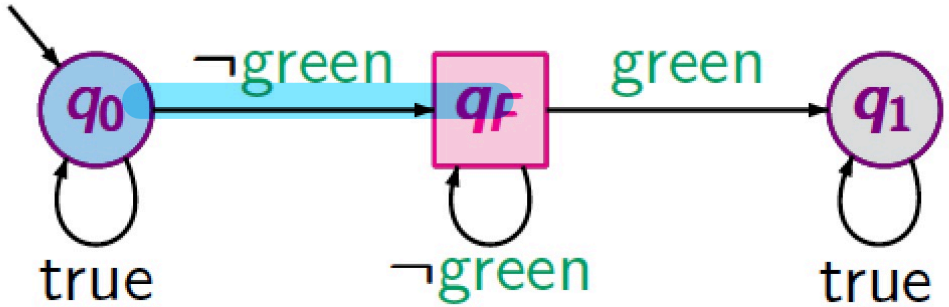
Example (1)

TS $\mathcal{T}$

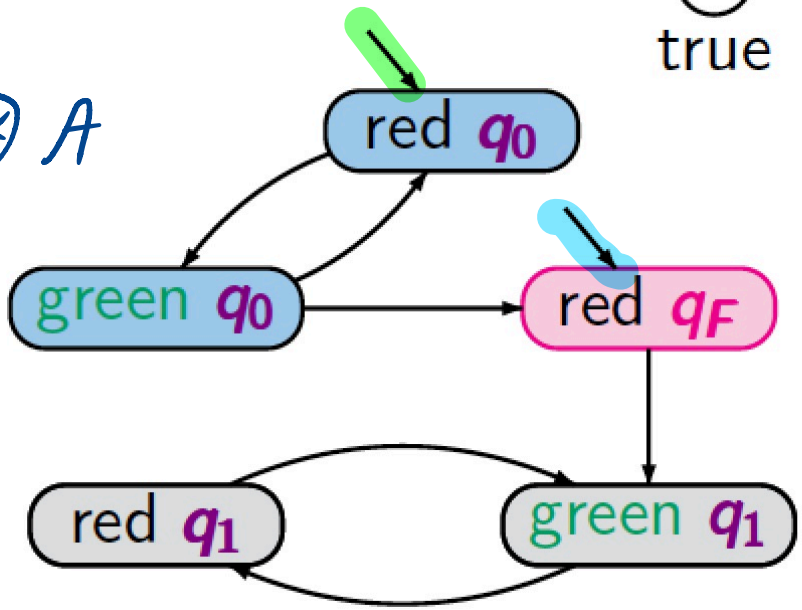


LT property: "infinitely often green"

NBA $\mathcal{A}$ for the complement
"from some moment on ¬green"



$\mathcal{T} \otimes \mathcal{A}$

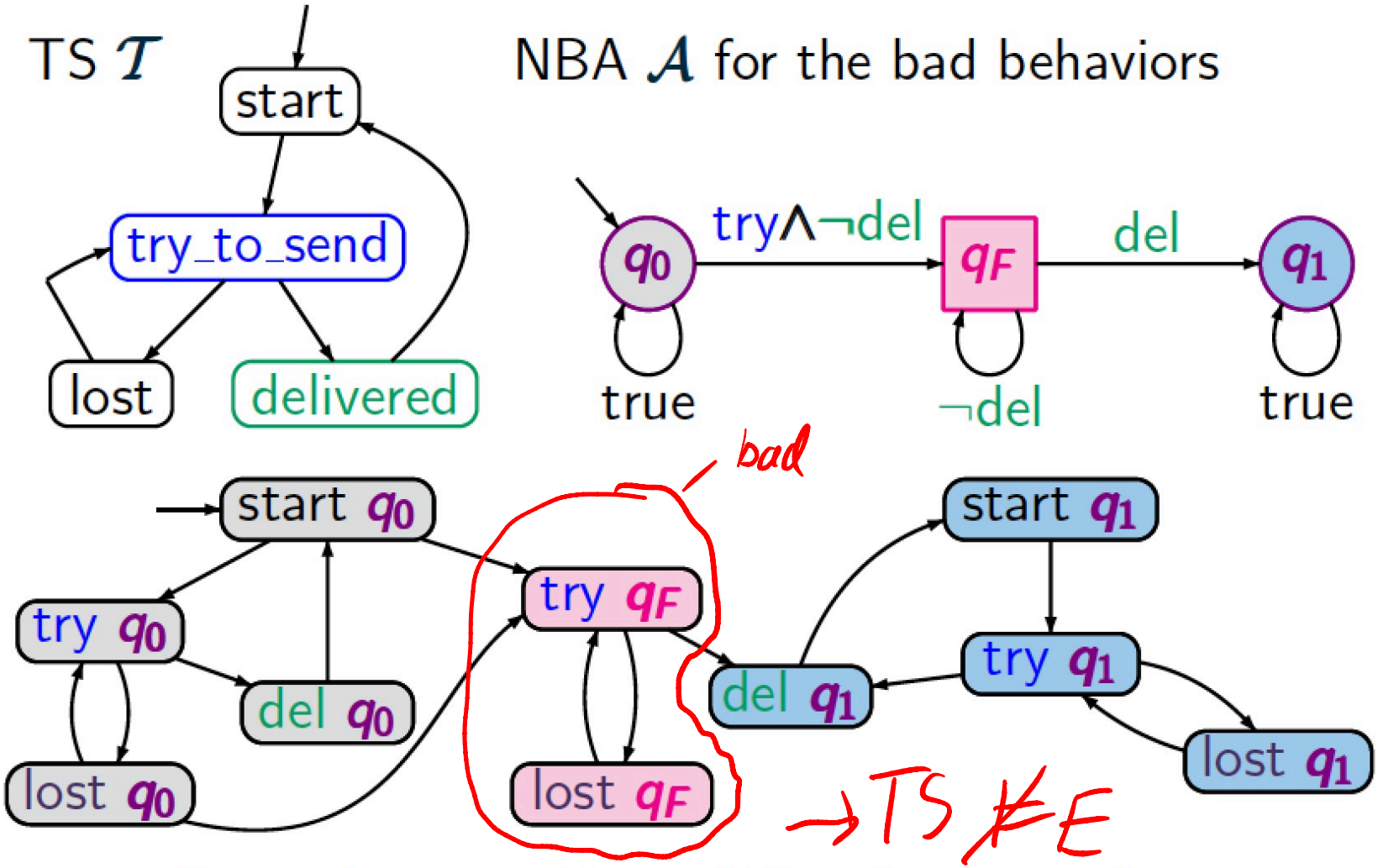


atomic propositions
$AP' = \{q_0, q_F, q_1\}$
obvious labeling function

$\mathcal{T} \otimes \mathcal{A} \models$
"eventually forever ¬F"

Example (2)

"Each attempt eventually succeeds"



set of atomic propositions $AP' = \{q_0, q_1, q_F\}$

Overview

Persistence Checking and Cycle Detection

Let

- TS be a finite transition system over AP without terminal states
- Φ a propositional formula over AP, and
- E the persistence property "eventually forever Φ"

$$TS \not\models E$$

if and only if

(1) $$\exists s \in \mathit{Reach}(TS).\ s \not\models \Phi \wedge s \text{ is on a cycle in } TS$$

if and only if

(2) $\exists$ a non-trivial reachable SCC C with $C \cap \{ s \in S \mid s \models \neg\Phi \} \neq \emptyset$

(maximal) strongly connected components

Persistence Checking

How to check for a reachable cycle containing a $\neg\Phi$-state?

Two linear-time algorithms:

- Alternative 1: (based on (2); non-trivial)
  - compute the maximal strongly connected components (SCCs) in TS
  - check whether some SCC is reachable from an initial state
  - ... that contains a $\neg\Phi$-state

Persistence Checking

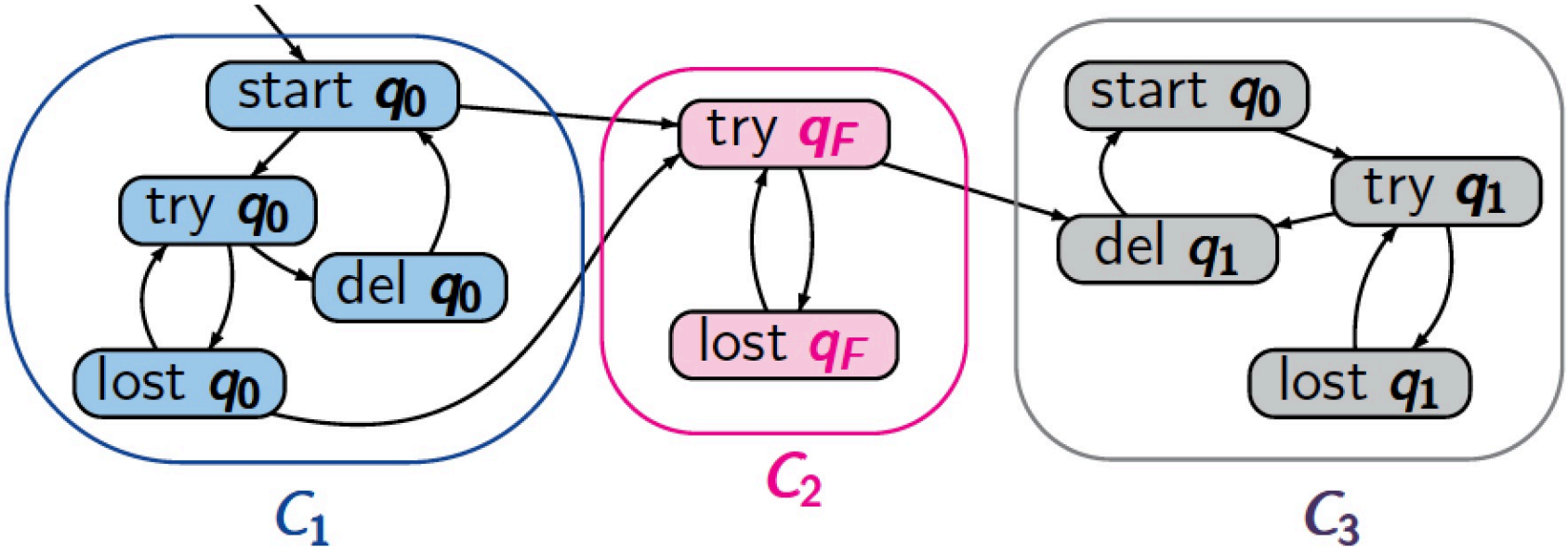
How to check for a reachable cycle containing a $\neg\Phi$-state?

Two linear-time algorithms:

- Alternative 1:
  - compute the maximal strongly connected components (SCCs) in TS
  - check whether some SCC is reachable from an initial state
  - ... that contains a $\neg\Phi$-state
- Alternative 2:
  - use a nested depth-first search
  - for each reachable $\neg\Phi$-state, check whether it belongs to a cycle
  - more adequate for on-the-fly verification algorithm
  - enables simple counterexample generation

based on (7)

Example SCC Algorithm



persistence property: "eventually forever $\neg q_F$"

3 reachable SCCs: C_1, C_2, C_3

C_2 non-trivial, and contains two states s with $s \not\models \neg q_F$

$\mathcal{T} \otimes \mathcal{A} \not\models$ "eventually forever $\neg q_F$"

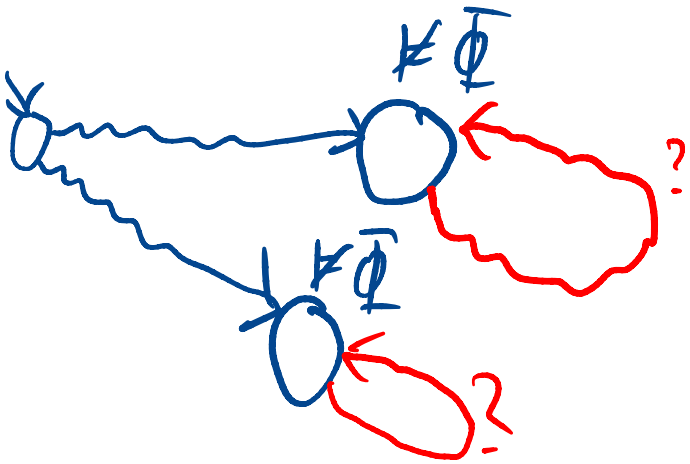
A Naive Two-Phase Depth First-Search

1. Determine all $\neg\Phi$-states that are reachable from some initial state
   this is performed by a standard depth-first search

A Naive Two-Phase Depth First-Search

1. Determine all $\neg\Phi$-states that are reachable from some initial state
   this is performed by a standard depth-first search

2. For each reachable $\neg\Phi$-state, check whether it belongs to a cycle
   - start a depth-first search in $\neg\Phi$-state s
   - to check whether s is reachable from itself



A Naive Two-Phase Depth First-Search

1. Determine all $\neg\Phi$-states that are reachable from some initial state
   this is performed by a standard depth-first search

2. For each reachable $\neg\Phi$-state, check whether it belongs to a cycle
   - start a depth-first search in $\neg\Phi$-state s
   - to check whether s is reachable from itself

- Time complexity naive algorithm: $\Theta(N \cdot (\underbrace{N+M}_{\text{cycle check}}))$
  - where N is the number of states and M the number of transitions
  - where it is assumed that checking Φ is in $O(1)$
  - states reachable via K distinct $\neg\Phi$-states are searched K times

Time complexity nested DFS: $\Theta(N \cdot M)$.

Nested Depth-First Search

- Idea: perform the two depth-first searches in an *interleaved* way
  - the outer DFS serves to encounter all reachable $\neg\Phi$-states
  - the inner DFS seeks for backward edges leading to a $\neg\Phi$-state

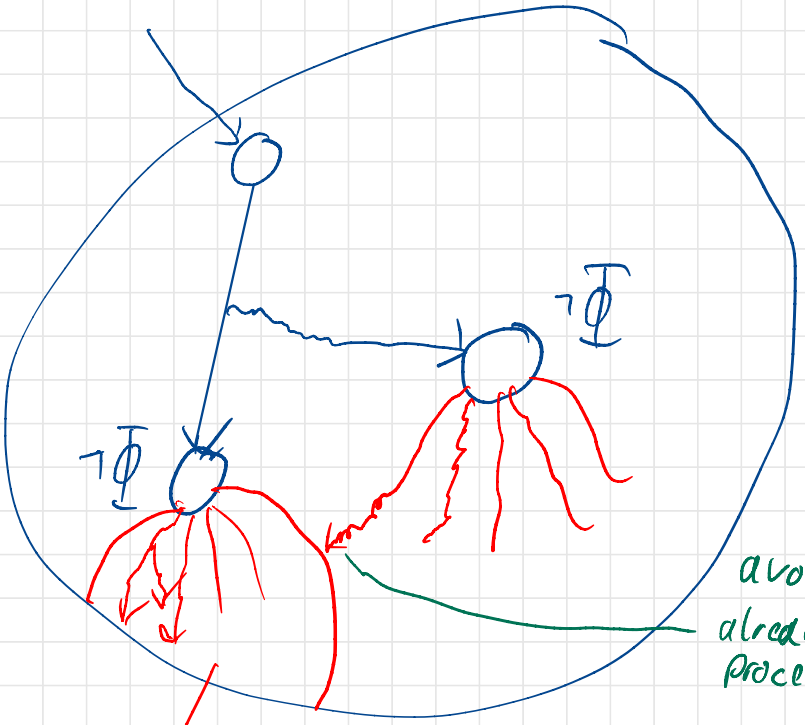
Nested Depth-First Search

- Idea: perform the two depth-first searches in an *interleaved* way
  - the outer DFS serves to encounter all reachable $\neg\Phi$-states
  - the inner DFS seeks for backward edges leading to a $\neg\Phi$-state

- Nested DFS
  - on full expansion of $\neg\Phi$-state s in the outer DFS, start inner DFS
  - in the inner DFS, visit all states reachable from s that have not been visited in an inner DFS yet
  - backward edge found?

    a cycle containing $\neg\Phi$-state s found
  - no backward edge found to s?

    continue the outer DFS (look for next $\neg\Phi$-state)


¬ϕ
¬ϕ
avoid traversing
already
processed states
traversed in inner dfs

Algorithm for "outer" DFS

LIFO

$U := \emptyset$; $\pi := \emptyset$; ← visiting set and stack for 1. DFS

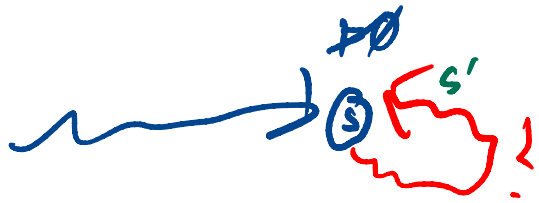
$V := \emptyset$; $\xi := \emptyset$; ← visiting set and stack for 2. DFS

```
WHILE S0 ⊈ U DO
    choose s0 ∈ S0 \ U; insert s0 in U; Push(π, s0);
    WHILE π ≠ ∅ DO
        s := Top(π);
        IF Post(s) ⊈ U
         THEN choose s' ∈ Post(s) \ U;
              insert s' in U; Push(π, s')
         ELSE Pop(π);
              IF s ⊭ φ and CYCLE_CHECK(s, V, ξ)
                THEN return "no" + reverse(π, ξ) FI
        FI
    OD OD
    return "yes"
```

TS ≠ E

counter example

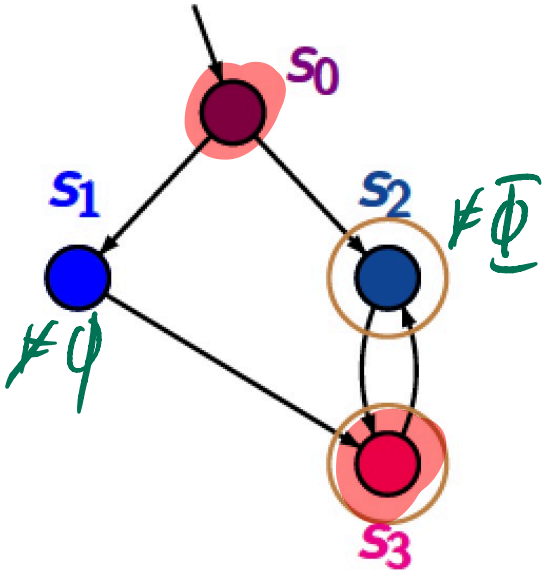
Algorithm for "inner" DFS




```
CYCLE_CHECK(s, V, ξ):
    Push(ξ, s); insert s in V;
    WHILE ξ ≠ ∅ DO
        s' := Top(ξ);
        IF s ∈ Post(s')
            THEN Push(ξ, s); return "true"
            ELSE IF Post(s') ⊈ V
                    THEN choose s'' ∈ Post(s') \ V;
                         insert s'' in V; Push(ξ, s'');
                    ELSE Pop(ξ)
                 FI
        FI
    OD
    return "false"
```

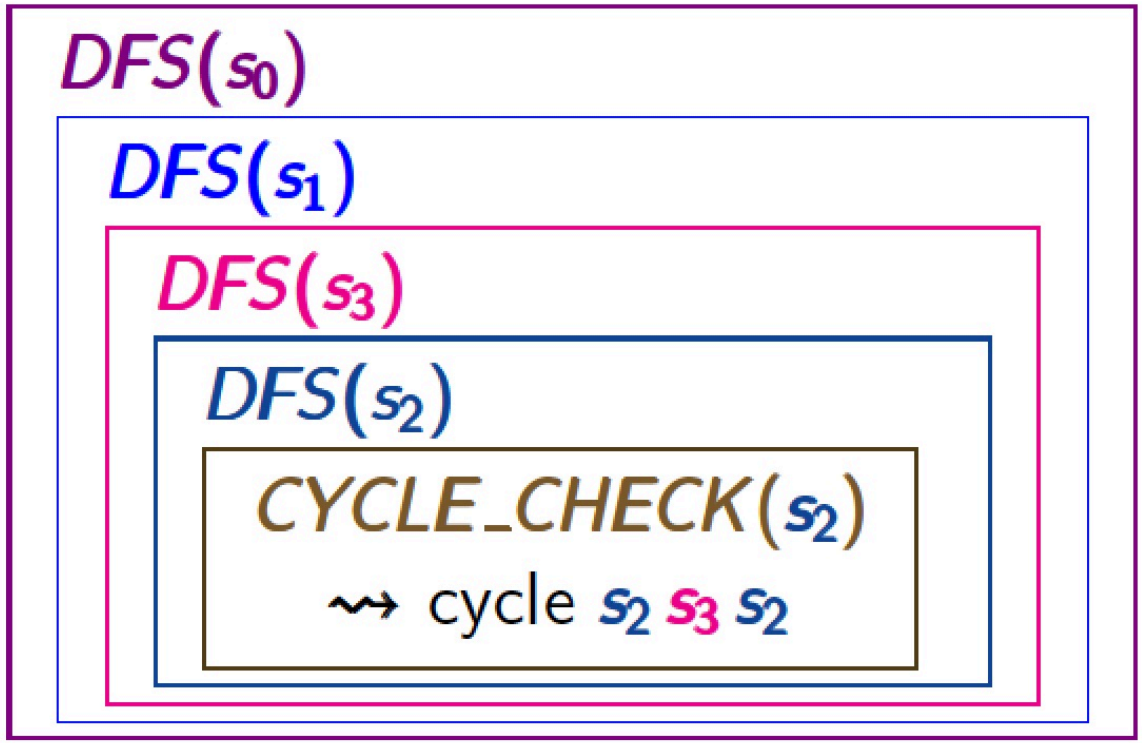
For counter example ⇝ there is a cycle

Nested DFS: Example



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$



$\mathcal{T} \not\models$ "eventually forever a"

↑
returns correct answer "**no**"

Correctness of Nested DFS

Let:

- TS be a finite transition system over AP without terminal states and
- E a persistence property.

Then:

The nested DFS algorithm yields "no" if and only if $TS \not\models E$.

The nested DFS algorithm yields "no" if and only if $TS \not\models E$.

"$\Leftarrow$" ✓

"$\Rightarrow$" key lemma is to show:

on invoking cycle_check(s),

then no cycle

to $t_1 \ldots t_k$ exists s.t.

s = t_0 ... s = t_k

$t_i \in V$ for some i

↑ set of processed states in previous calls of cycle_check

Time Complexity

The worst-case time complexity of nested DFS is in

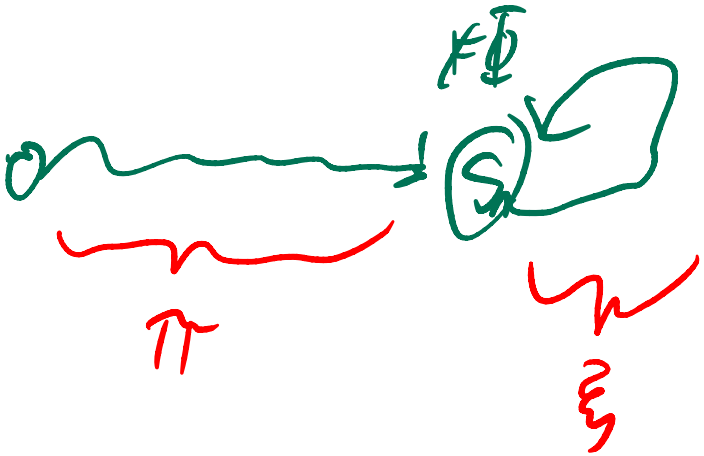
$$\Theta(N+M)$$

where N is # states in TS, and M is # transitions in TS.

Counterexamples

A counterexample to $TS \models$ eventually forever Φ is an initial path fragment of the form

$$\underbrace{s_0}_{\in I} \ldots s_{n-1} \underbrace{s_n}_{\models \neg\Phi} s_{n+1} \ldots s_{n+m-1} \underbrace{s_n}_{\models \neg\Phi} \quad \text{for } m > 0.$$



Counterexamples

A counterexample to $TS \models$ eventually forever Φ is an initial path fragment of the form

$$\underbrace{s_0}_{\in I} \ldots s_{n-1} \; \underbrace{s_n}_{\models \neg\Phi} \; s_{n+1} \ldots s_{n+m-1} \; \underbrace{s_n}_{\models \neg\Phi} \qquad \text{for } m > 0.$$

Using nested depth-first search:

- Counterexample generation: use the DFS stacks
  - stack π_{out} for the outer DFS = path fragment $s_n\, s_{n-1} \ldots s_0$
  - stack π_{in} for the inner DFS = a cycle from state $s_n\, s_{n+m-1} \ldots s_n$
  - counterexample = *reverse* $(\pi_{in}.\pi_{out})$

Overview

Summary

- Checking a regular safety property E = checking invariant on product
  - with an NFA $\mathfrak{A}$ for the bad prefixes of E
  - "never reach an accept state of $\mathfrak{A}$"
- Checking ω-regular property E = checking persistence on a product
  - with an NBA for the complement of E
  - "eventually forever no accept state of $\mathfrak{A}$"
- Persistence checking is solvable in linear time by a nested DFS
- Nested DFS
  = a DFS for reachable $\neg\Phi$-states + a DFS for cycle detection

Next Lecture

Thursday May 5, 12:30