Overview

Verifying ω-regular properties

given: finite transition system $\mathcal{T}$
ω-regular property E

question: does $\mathcal{T} \models E$ hold ?

Verifying ω-regular properties

given: finite transition system $\mathcal{T}$
ω-regular property E

question: does $\mathcal{T} \models E$ hold ?

(1) construct an **NBA** $\mathcal{A}$ for the bad behaviors, i.e.,
$\mathcal{L}_\omega(\mathcal{A}) = \left(2^{AP}\right)^\omega \setminus E$

Verifying ω-regular properties

given: finite transition system $\mathcal{T}$
ω-regular property E

question: does $\mathcal{T} \models E$ hold ?

(1) construct an **NBA** $\mathcal{A}$ for the bad behaviors, i.e.,
$\mathcal{L}_\omega(\mathcal{A}) = (2^{AP})^\omega \setminus E$

(2) check whether $\mathit{Traces}(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

Verifying ω-regular properties

given: finite transition system $\mathcal{T}$
ω-regular property E

question: does $\mathcal{T} \models E$ hold ?

(1) construct an **NBA** $\mathcal{A}$ for the bad behaviors, i.e., $\mathcal{L}_\omega(\mathcal{A}) = (2^{AP})^\omega \setminus E$

(2) check whether $\mathit{Traces}(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

(3) build the product transition system $\mathcal{T} \otimes \mathcal{A}$ and check whether

$\mathcal{T} \otimes \mathcal{A} \models$ "never acceptance condition of $\mathcal{A}$"

Verifying ω-regular properties

given: finite transition system $\mathcal{T}$
ω-regular property E

question: does $\mathcal{T} \models E$ hold ?

(1) construct an **NBA** $\mathcal{A}$ for the bad behaviors, i.e., $\mathcal{L}_\omega(\mathcal{A}) = (2^{AP})^\omega \setminus E$

(2) check whether $\mathit{Traces}(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

(3) build the product transition system $\mathcal{T} \otimes \mathcal{A}$ and check whether

$\mathcal{T} \otimes \mathcal{A} \models$ "never acceptance condition of $\mathcal{A}$"

requires techniques for checking
persistence properties in finite TS

Persistence property

Persistence property

Let E be an LT-property, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$

E is called a persistence property if there exists a propositional formula Φ over AP such that

$$E = \left\{ \text{set of all infinite words } A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} \text{ s.t. } \overset{\infty}{\forall}\, i \geq 0.\ A_i \models \Phi \right\}$$

$\overset{\infty}{\forall}\, i{\geq}0.\ \ldots = \exists j{\geq}0\ \forall i{\geq}j.\ \ldots$ "for all but finitely many"

Persistence property

Let E be an LT-property, i.e., $E \subseteq (2^{AP})^{\omega}$

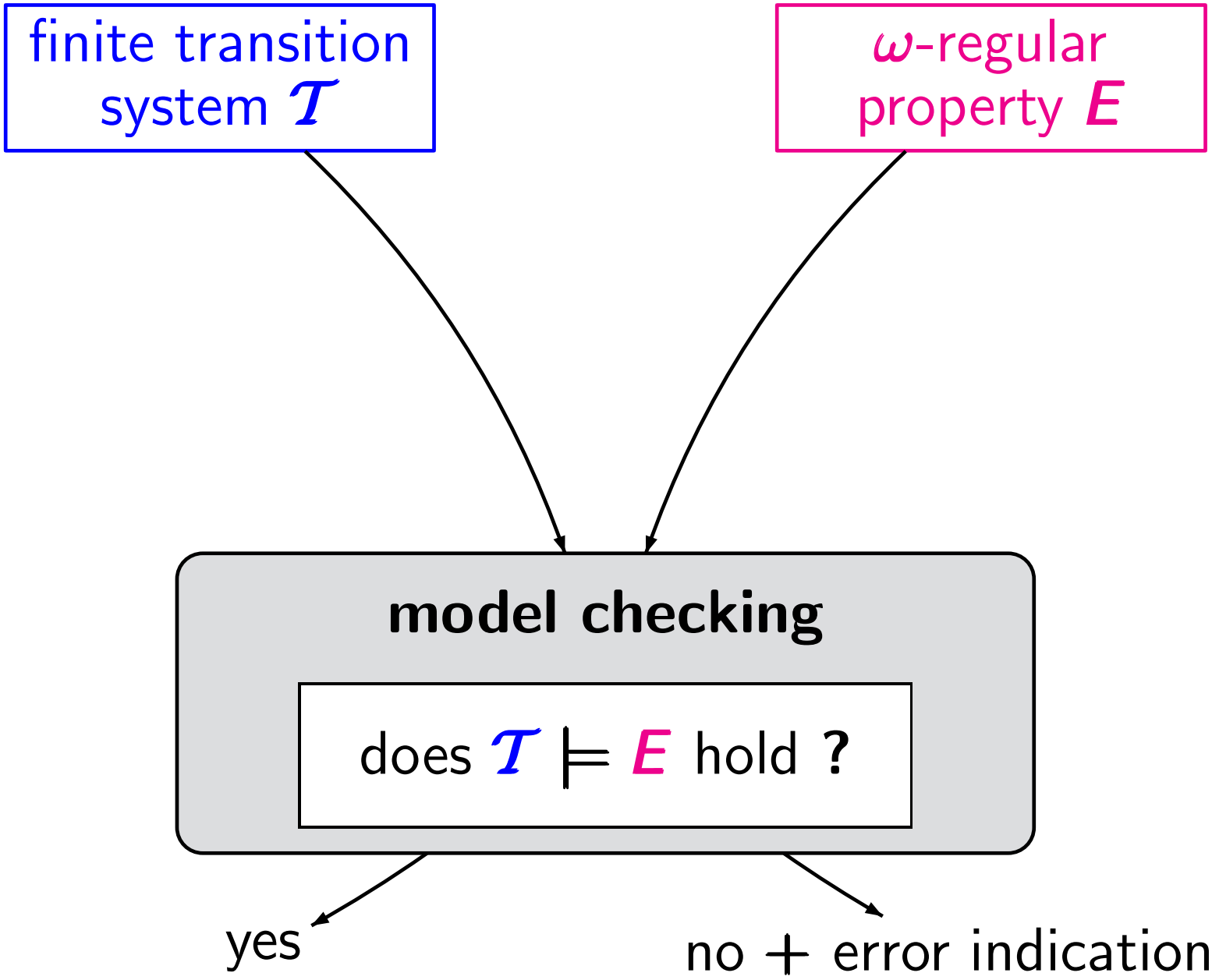
E is called a persistence property if there exists a propositional formula Φ over AP such that

$$E = \left\{ \text{set of all infinite words } A_0 A_1 A_2 \ldots \in (2^{AP})^{\omega} \text{ s.t. } \overset{\infty}{\forall}\, i \geq 0.\ A_i \models \Phi \right\}$$

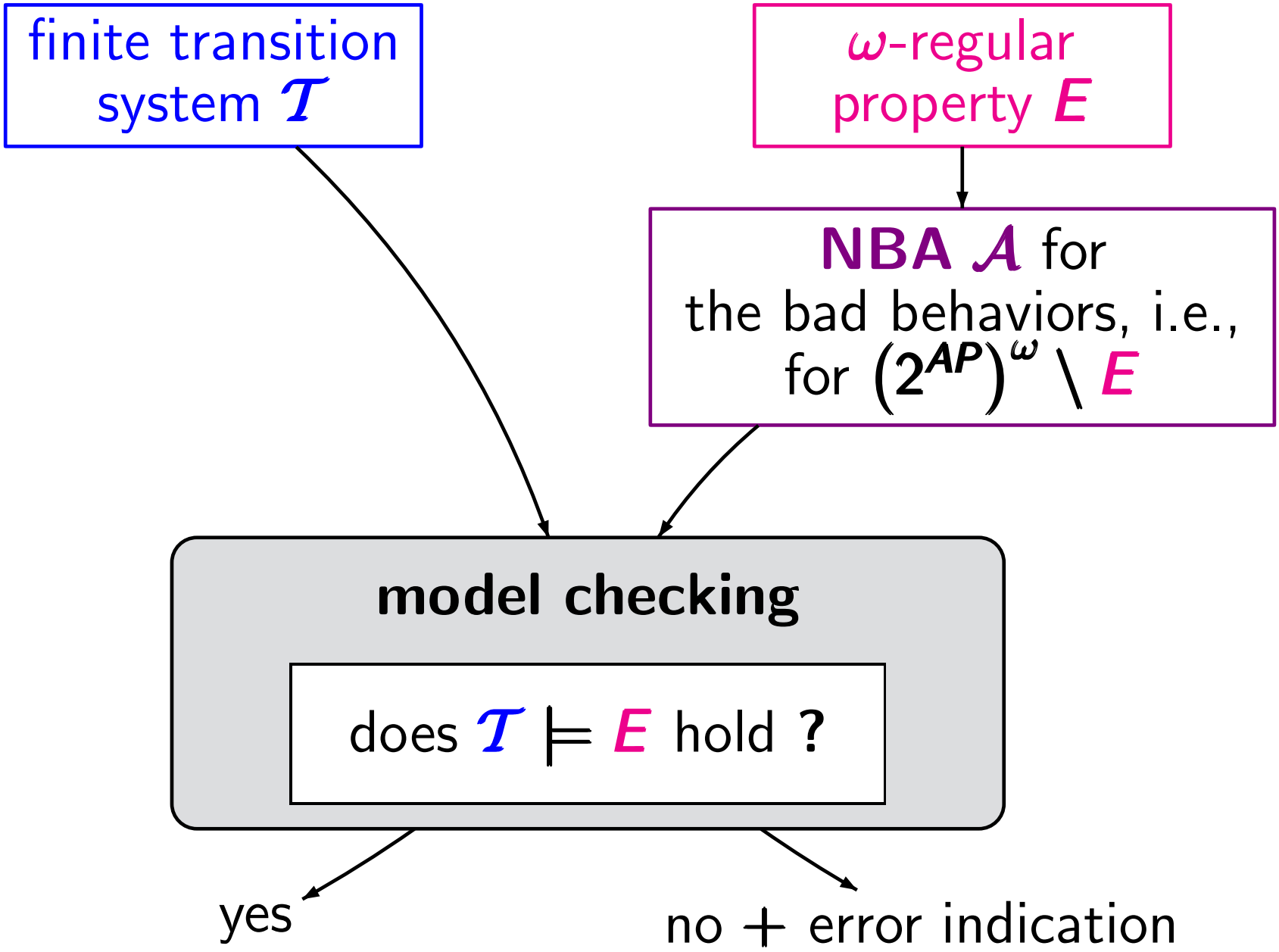
"from some moment on Φ"
"eventually forever Φ"

$\overset{\infty}{\forall}\, i{\geq}0.\ \ldots = \exists j{\geq}0\ \forall i{\geq}j.\ \ldots$ "for all but finitely many"

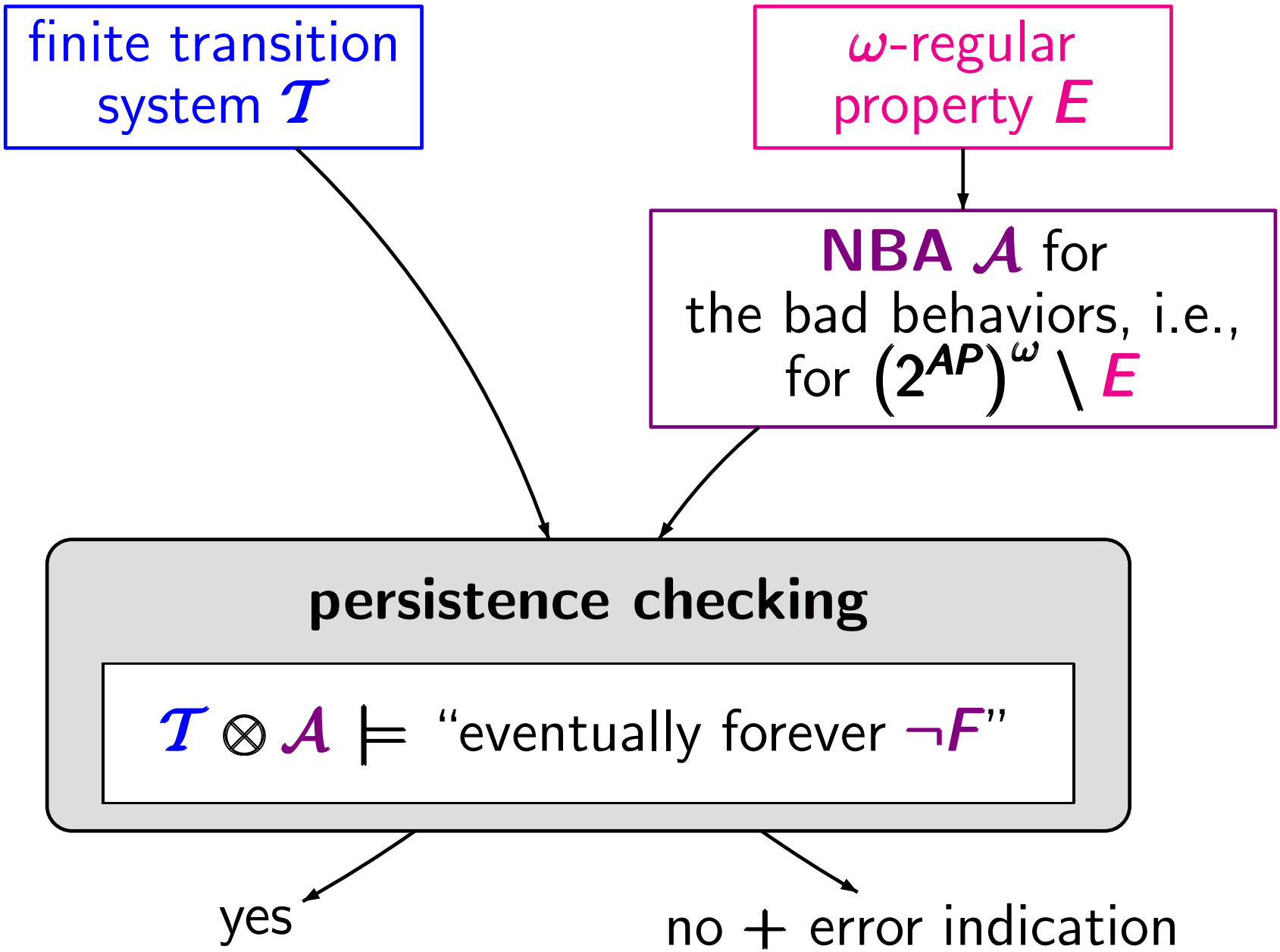
Checking ω-regular properties



Checking ω-regular properties



Checking ω-regular properties



Recall: product of a TS and an NFA

finite transition system
$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$

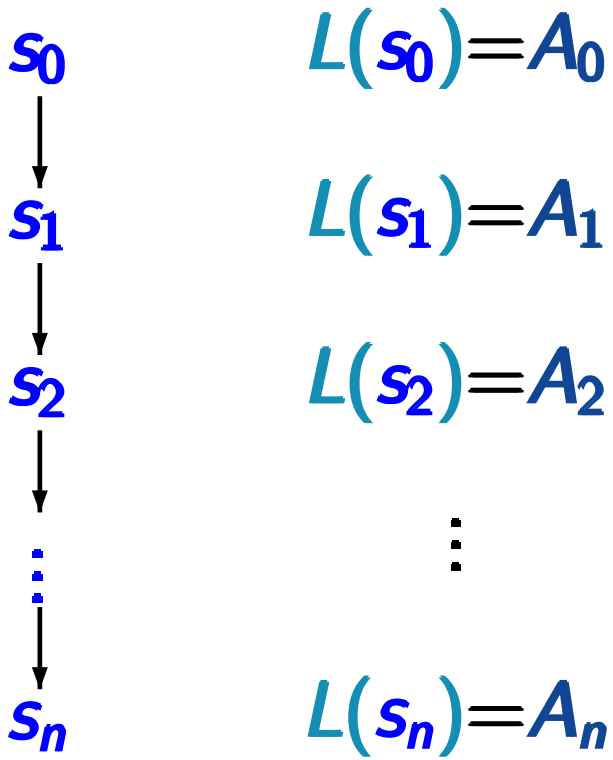
NFA for bad prefixes
$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$

path fragment $\widehat{\pi}$

Recall: product of a TS and an NFA

finite transition system
$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$

NFA for bad prefixes
$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$



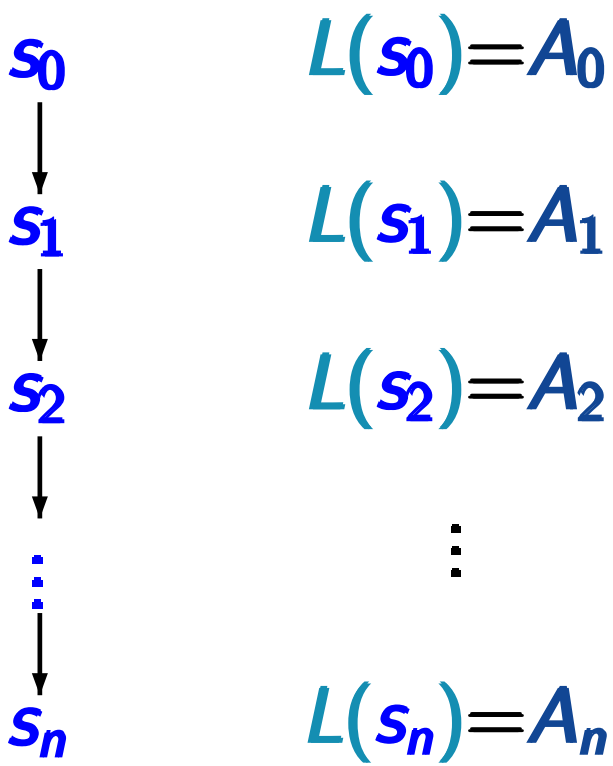
path fragment $\widehat{\pi}$

trace

Recall: product of a TS and an NFA

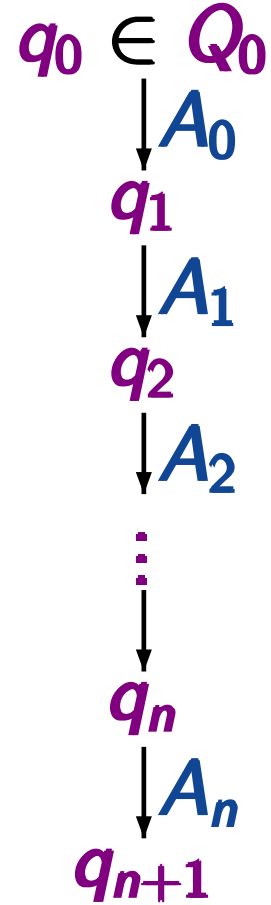
finite transition system
$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$

NFA for bad prefixes
$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$



path fragment $\widehat{\pi}$

trace

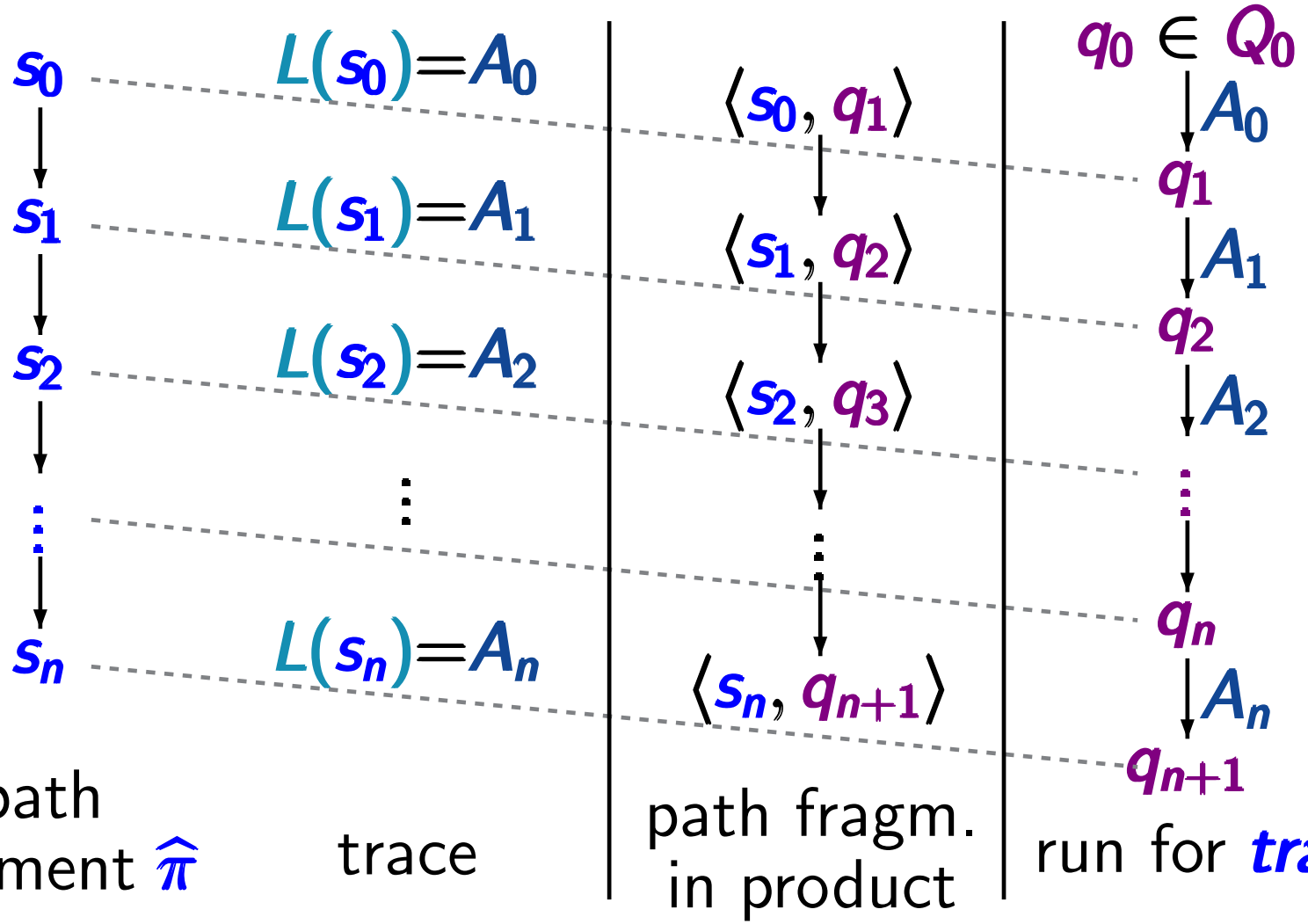


run for $trace(\widehat{\pi})$

Recall: product of a TS and an NFA

finite transition system
$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$

NFA for bad prefixes
$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$



Product transition system

recall: definition of the product of a **TS** and **NFA**

Product transition system

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ transition system

$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ NFA

product-TS $\mathcal{T} \otimes \mathcal{A} \stackrel{\text{def}}{=} (S \times Q, Act, \longrightarrow', S_0', AP', L')$

Product transition system

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ transition system

$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ NFA

product-TS $\mathcal{T} \otimes \mathcal{A} \stackrel{\text{def}}{=} (S \times Q, Act, \longrightarrow', S_0', AP', L')$

$$\frac{s \xrightarrow{\alpha} s' \quad \wedge \quad q' \in \delta(q, L(s'))}{\langle s, q \rangle \xrightarrow{\alpha}' \langle s', q' \rangle}$$

initial states: $S_0' = \{ \langle s_0, q \rangle : s_0 \in S_0, q \in \delta(Q_0, L(s_0)) \}$

Product transition system

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ transition system

$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ NFA

product-TS $\mathcal{T} \otimes \mathcal{A} \stackrel{\text{def}}{=} (S \times Q, Act, \longrightarrow', S_0', AP', L')$

$$\frac{s \xrightarrow{\alpha} s' \quad \wedge \quad q' \in \delta(q, L(s'))}{\langle s, q \rangle \xrightarrow{\alpha}{}' \langle s', q' \rangle}$$

initial states: $S_0' = \{ \langle s_0, q \rangle : s_0 \in S_0, q \in \delta(Q_0, L(s_0)) \}$

set of atomic propositions: $AP' = Q$

labeling function: $L'(\langle s, q \rangle) = \{q\}$

Product transition system

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ transition system

$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ NFA $\leftarrow$ same definition for **NBA**

product-TS $\mathcal{T} \otimes \mathcal{A} \stackrel{\text{def}}{=} (S \times Q, Act, \longrightarrow', S'_0, AP', L')$

$$\frac{s \xrightarrow{\alpha} s' \quad \wedge \quad q' \in \delta(q, L(s'))}{\langle s, q \rangle \xrightarrow{\alpha}' \langle s', q' \rangle}$$

initial states: $S'_0 = \{ \langle s_0, q \rangle : s_0 \in S_0, q \in \delta(Q_0, L(s_0)) \}$

set of atomic propositions: $AP' = Q$

labeling function: $L'(\langle s, q \rangle) = \{q\}$

Product of a TS and NBA

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ transition system

$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ **NFA** or **NBA**

product-TS $\mathcal{T} \otimes \mathcal{A} \stackrel{\text{def}}{=} (S \times Q, Act, \longrightarrow', S'_0, AP', L')$

$$\frac{s \xrightarrow{\alpha} s' \quad \wedge \quad q' \in \delta(q, L(s'))}{\langle s, q \rangle \xrightarrow{\alpha}' \langle s', q' \rangle}$$

initial states: $S'_0 = \{ \langle s_0, q \rangle : s_0 \in S_0, q \in \delta(Q_0, L(s_0)) \}$

set of atomic propositions: $AP' = Q$

labeling function: $L'(\langle s, q \rangle) = \{q\}$

ω-regular model checking

given:	finite TS $\mathcal{T}$
	ω-regular LT property E
question:	does $\mathcal{T} \models E$ hold **?**

ω-regular model checking

given:	finite TS $\mathcal{T}$
	ω-regular LT property E
question:	does $\mathcal{T} \models E$ hold ?

algorithm uses an **NBA** for the bad behaviors for E

ω-regular model checking

given:	finite TS $\mathcal{T}$
	ω-regular LT property E
question:	does $\mathcal{T} \models E$ hold ?

algorithm uses an **NBA** for the bad behaviors for E

relies on a reduction to the persistence checking problem

ω-regular model checking

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ finite transition system without terminal states

$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ non-blocking NBA

representing the bad behaviors of an ω-regular LT-property E

ω-regular model checking

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ finite transition system without terminal states

$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ non-blocking NBA

representing the bad behaviors of an ω-regular LT-property E, i.e., $\mathcal{L}_\omega(\mathcal{A}) = \left(2^{AP}\right)^\omega \setminus E$

ω-regular model checking

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ finite transition system without terminal states

$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ non-blocking NBA

representing the bad behaviors of an ω-regular LT-property E, i.e., $\mathcal{L}_\omega(\mathcal{A}) = \left(2^{AP}\right)^\omega \setminus E$

The following statements are equivalent:

(1) $\mathcal{T} \models E$

(2) $Traces(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

ω-regular model checking

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ finite transition system without terminal states

$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ non-blocking NBA

representing the bad behaviors of an ω-regular LT-property E, i.e., $\mathcal{L}_\omega(\mathcal{A}) = (2^{AP})^\omega \setminus E$

The following statements are equivalent:

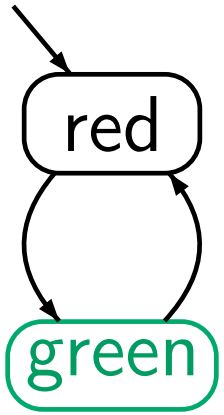
(1) $\mathcal{T} \models E$

(2) $Traces(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

(3) $\mathcal{T} \otimes \mathcal{A} \models$ "eventually forever $\neg F$"

Example: ω-regular model checking

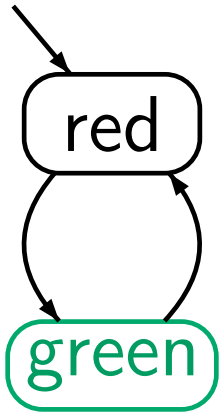
TS $\mathcal{T}$



LT property: "infinitely often green"

Example: ω-regular model checking

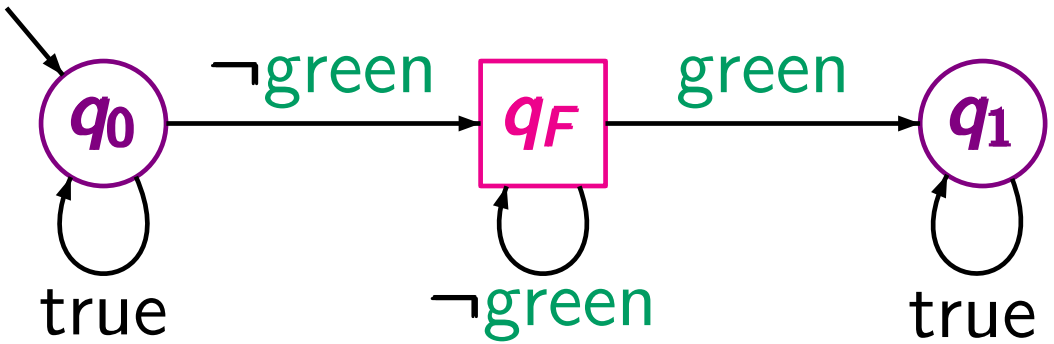
TS $\mathcal{T}$



LT property: "infinitely often green"

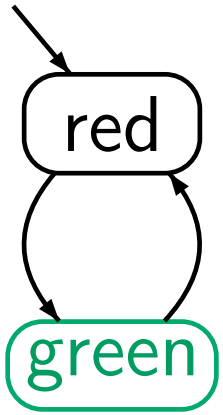
NBA $\mathcal{A}$ for the complement

"from some moment on ¬green"



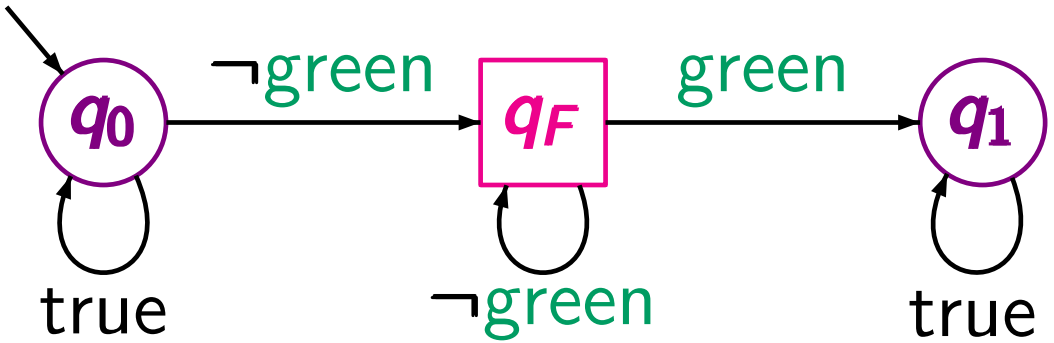
Example: ω-regular model checking

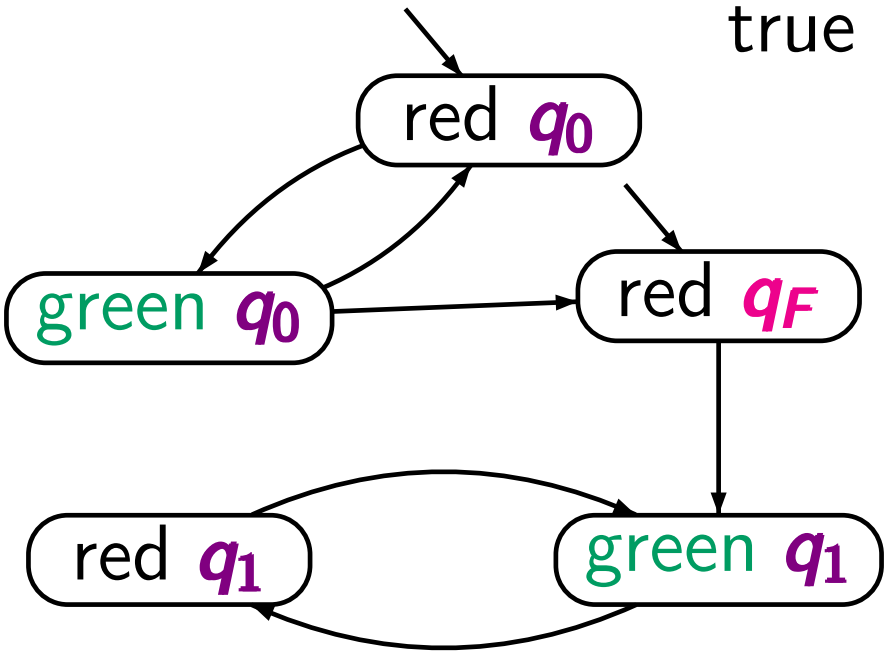
TS $\mathcal{T}$



LT property: "infinitely often green"

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

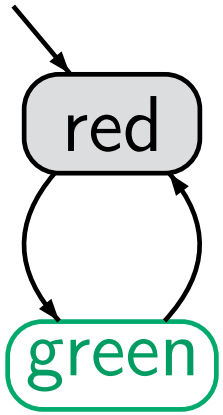




reachable fragment of the product TS $\mathcal{T} \otimes \mathcal{A}$

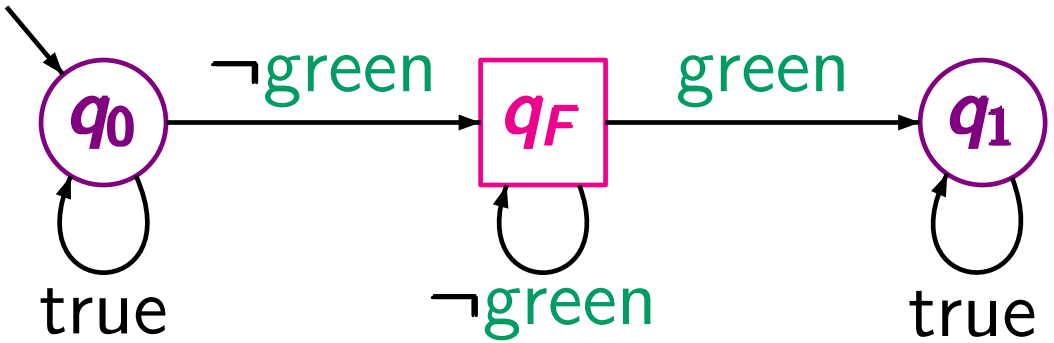
Example: ω-regular model checking

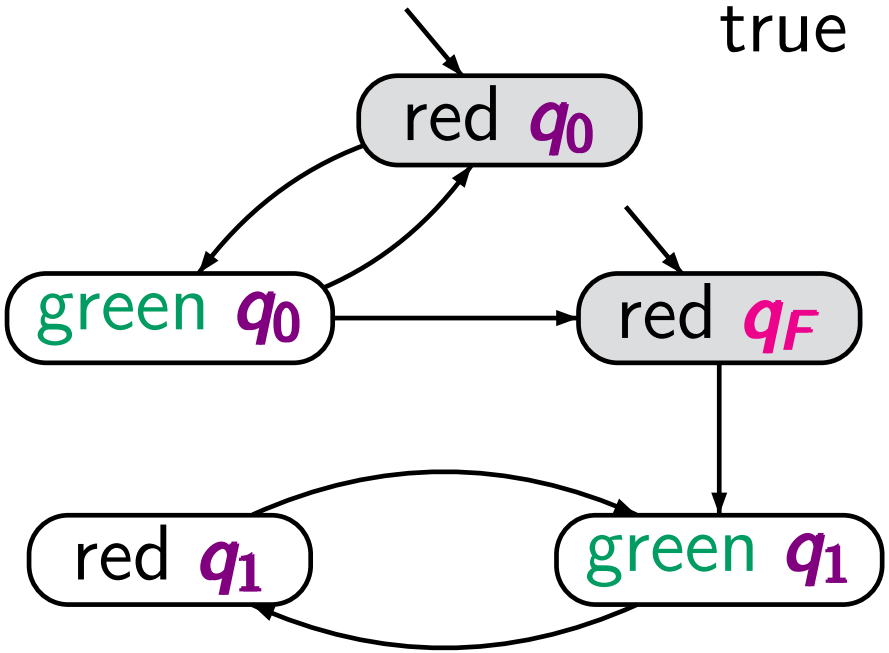
TS $\mathcal{T}$



LT property: "infinitely often green"

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





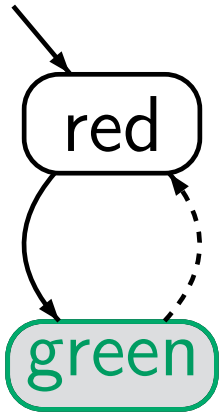
initial states:

$\langle \text{red}, q \rangle$ where

$$q \in \delta(q_0, L(\text{red})) = \delta(q_0, \varnothing) = \{q_0, q_F\}$$

Example: ω-regular model checking

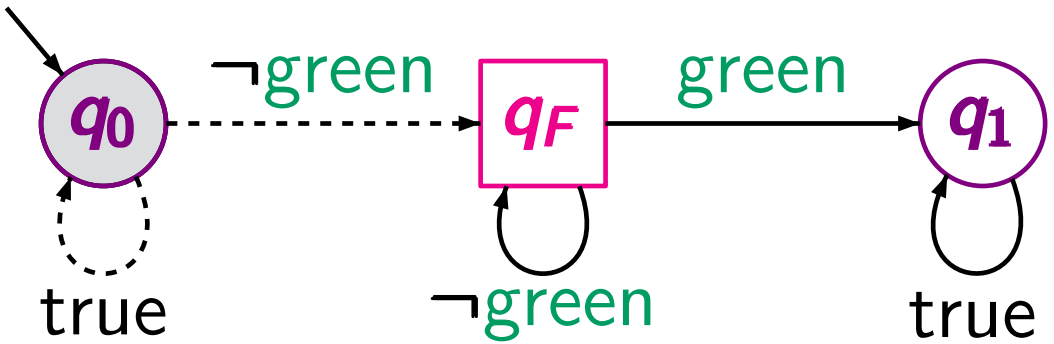
TS $\mathcal{T}$

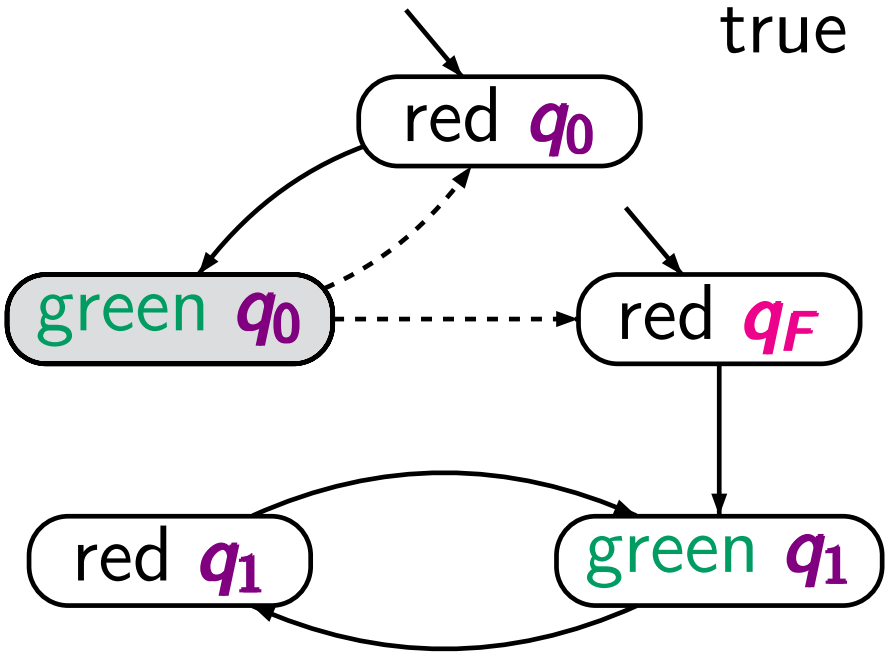


LT property: "infinitely often green"

NBA $\mathcal{A}$ for the complement

"from some moment on ¬green"





transition

$\langle \text{green}, q_0 \rangle \longrightarrow \langle \text{red}, q \rangle$

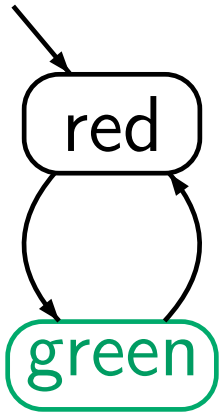
$q \in \delta(q_0, L(\text{red}))$

$= \delta(q_0, \varnothing)$

$= \{q_0, q_F\}$

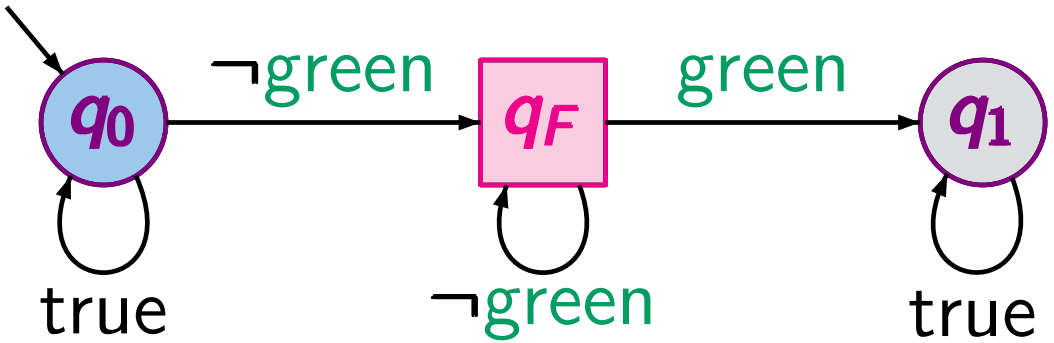
Example: ω-regular model checking

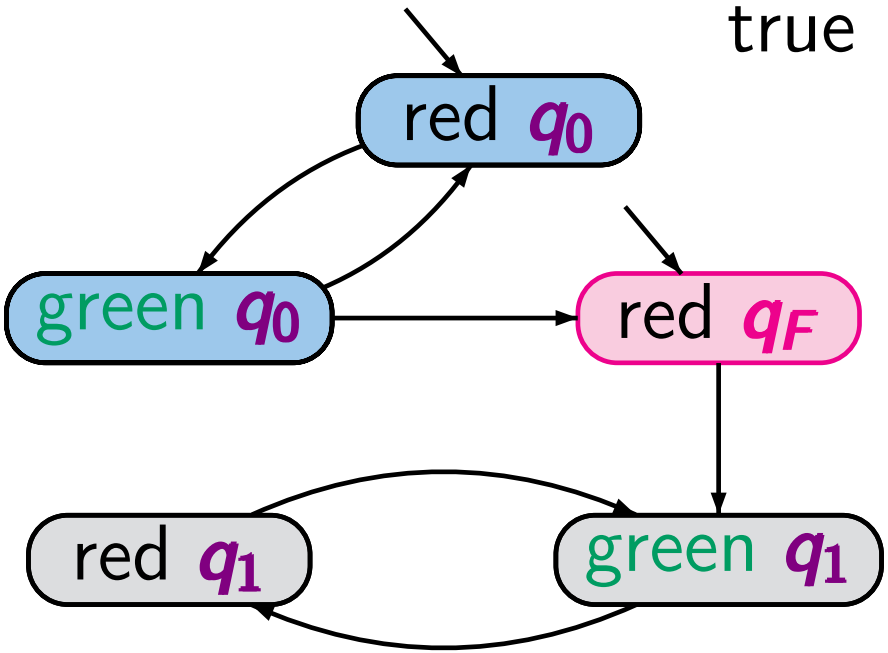
TS $\mathcal{T}$



LT property: "infinitely often green"

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

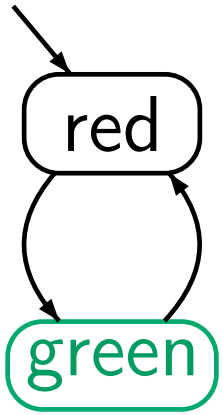




atomic propositions
$AP' = \{q_0, q_F, q_1\}$
obvious labeling function

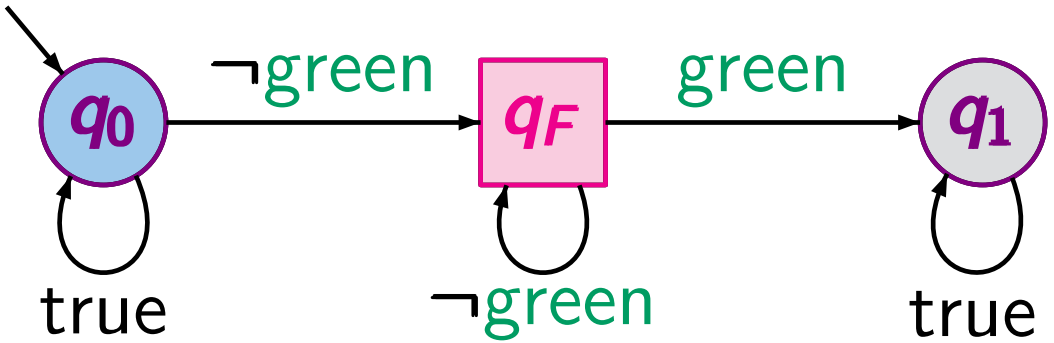
Example: ω-regular model checking

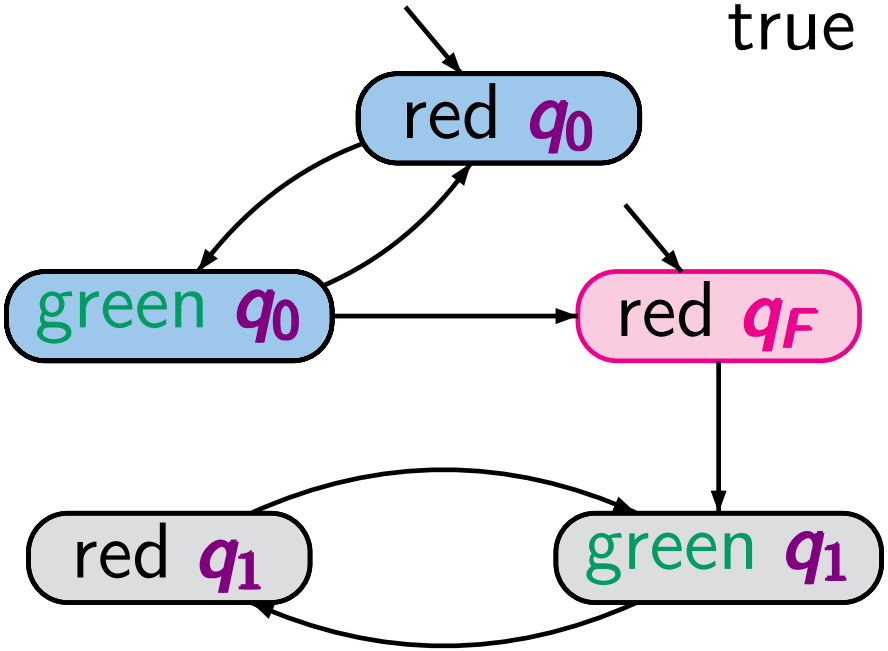
TS $\mathcal{T}$



LT property: "infinitely often green"

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

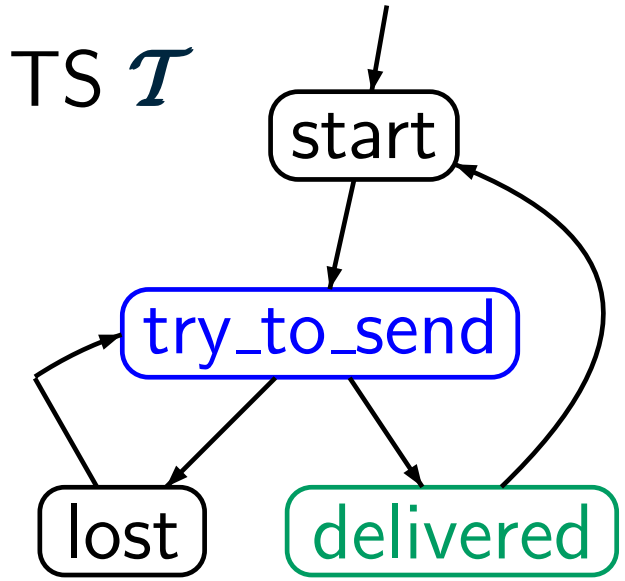




atomic propositions
$AP' = \{q_0, q_F, q_1\}$
obvious labeling function

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

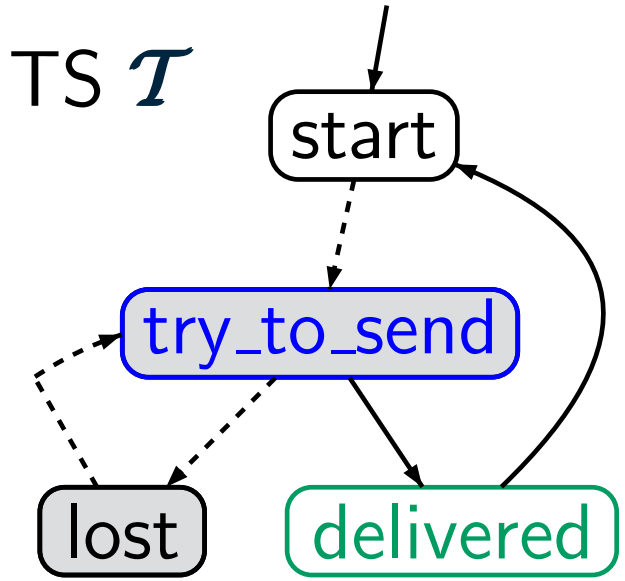
Example: ω-regular model checking



ω-regular LT property E:

"each (repeatedly) sent message will eventually be delivered"

Example: ω-regular model checking



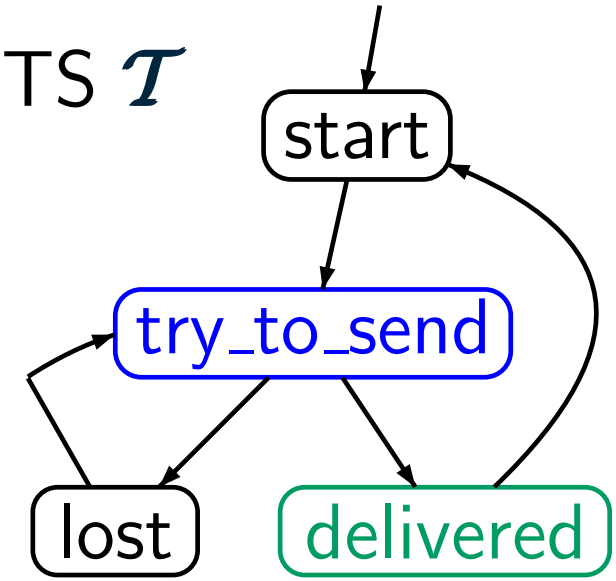
ω-regular LT property E:

"each (repeatedly) sent message will eventually be delivered"

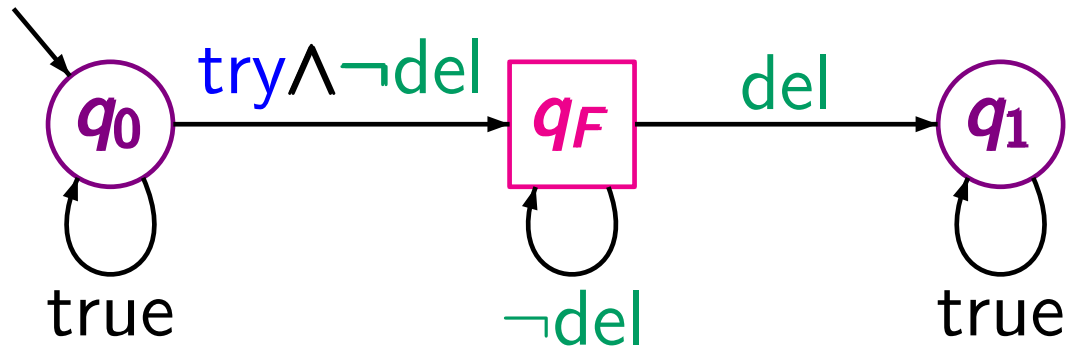
$\mathcal{T} \not\models E$

Example: ω-regular model checking

TS $\mathcal{T}$



NBA $\mathcal{A}$ for the bad behaviors



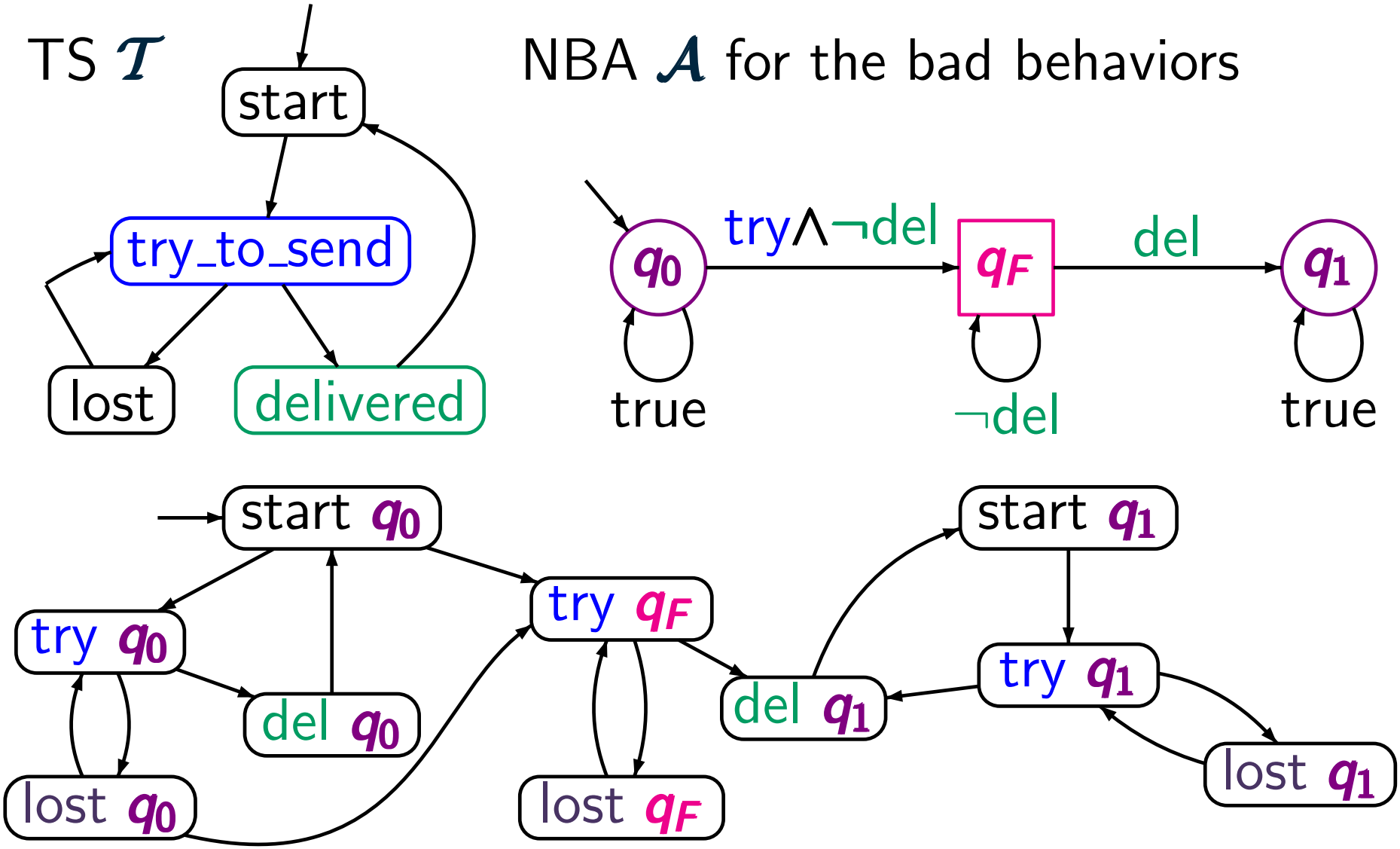
ω-regular LT property E:

"each (repeatedly) sent message will eventually be delivered"

complement of E, i.e., LT property for the bad behaviors:

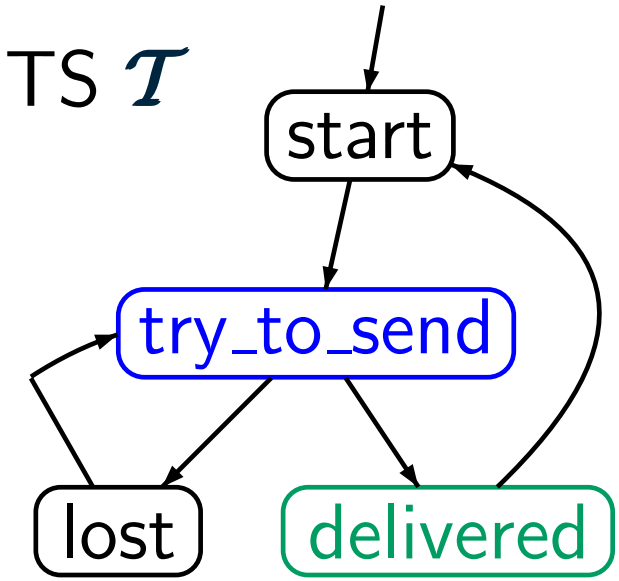
"never delivered after some trial"

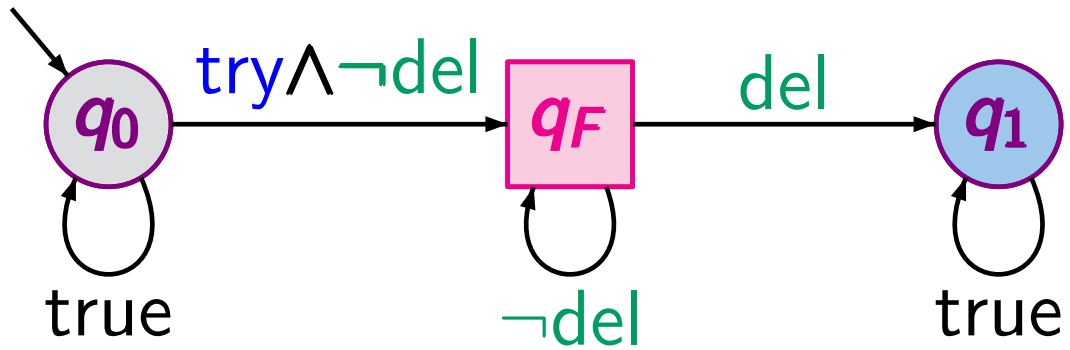
Example: ω-regular model checking

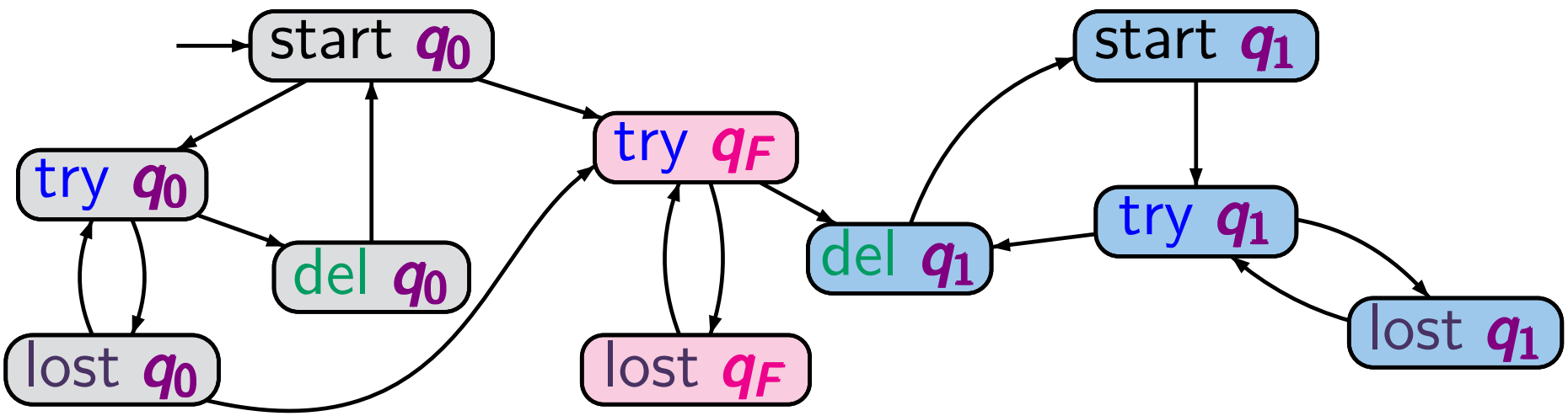


reachable fragment of the product-TS

Example: ω-regular model checking

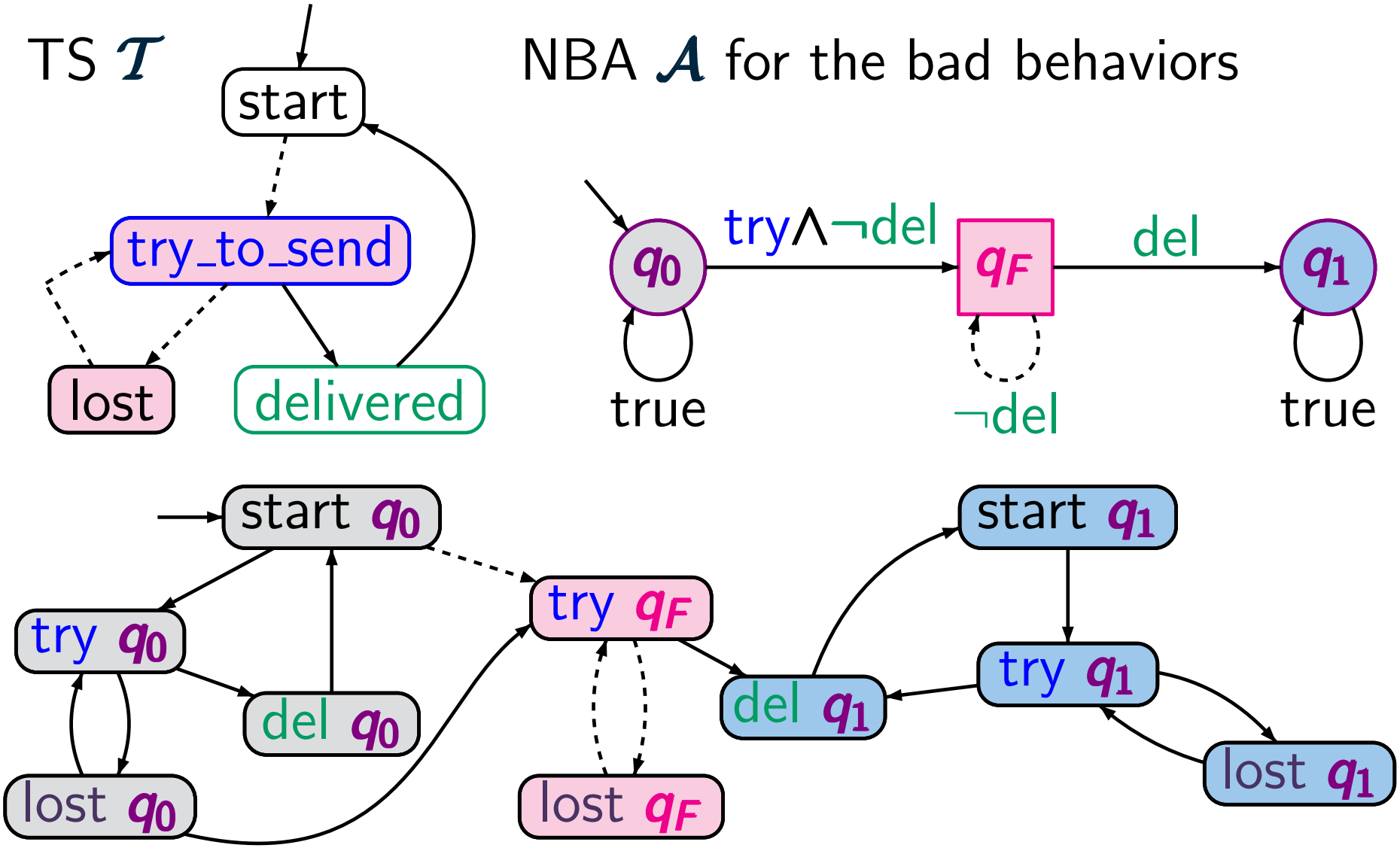






set of atomic propositions $AP' = \{q_0, q_1, q_F\}$

Example: ω-regular model checking



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

Checking safety and ω-regular properties

Checking safety and ω-regular properties

for regular safety property E

$$\mathcal{T} \models E$$

iff $\mathit{Traces}_{\mathit{fin}}(\mathcal{T}) \cap \mathit{BadPref} = \varnothing$

Checking safety and ω-regular properties

for regular safety property E

$$\mathcal{T} \models E$$

iff $\mathit{Traces}_{\mathit{fin}}(\mathcal{T}) \cap \mathit{BadPref} = \varnothing$

for ω-regular property E

$$\mathcal{T} \models E$$

iff $\mathit{Traces}(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

$\mathcal{A}$ is an **NBA** for the bad behaviors of E

Checking safety and ω-regular properties

for regular safety property E

$$\mathcal{T} \models E$$

iff $\mathit{Traces}_{\mathit{fin}}(\mathcal{T}) \cap \mathcal{L}(\mathcal{A}) = \varnothing$

$\mathcal{A}$ is an **NFA** for the bad prefixes of E

for ω-regular property E

$$\mathcal{T} \models E$$

iff $\mathit{Traces}(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

$\mathcal{A}$ is an **NBA** for the bad behaviors of E

Checking safety and ω-regular properties

for regular safety property E

$\mathcal{T} \models E$

iff $\mathit{Traces}_{\mathit{fin}}(\mathcal{T}) \cap \mathcal{L}(\mathcal{A}) = \varnothing$

iff $\mathcal{T} \otimes \mathcal{A} \models$ "forever $\neg F$"

$\mathcal{A}$ is an **NFA** for the bad prefixes of E

for ω-regular property E

$\mathcal{T} \models E$

iff $\mathit{Traces}(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

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

$\mathcal{A}$ is an **NBA** for the bad behaviors of E

F = set of final states in $\mathcal{A}$

Checking safety and ω-regular properties

for regular safety property E

$\mathcal{T} \models E$

iff $\mathit{Traces}_{\mathit{fin}}(\mathcal{T}) \cap \mathcal{L}(\mathcal{A}) = \varnothing$

iff $\mathcal{T} \otimes \mathcal{A} \models$ "forever $\neg F$"

invariant checking

for ω-regular property E

$\mathcal{T} \models E$

iff $\mathit{Traces}(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

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

persistence checking

F = set of final states in $\mathcal{A}$

Persistence checking

given: finite transition system $\mathcal{T}$ over AP
persistence condition $a \in AP$

question: does $\mathcal{T} \models$ "eventually forever a" hold **?**

Persistence checking

given: finite transition system $\mathcal{T}$ over AP
persistence condition $a \in AP$

question: does $\mathcal{T} \models$ "eventually forever a" hold ?

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

iff there is a path $s_0\, s_1\, s_2\, s_3 \ldots$ in $\mathcal{T}$ s.t.
$s_i \not\models a$ for infinitely many $i \geq 0$

Persistence checking

given: finite transition system $\mathcal{T}$ over AP
persistence condition $a \in AP$

question: does $\mathcal{T} \models$ "eventually forever a" hold ?

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

iff there is a path $s_0\, s_1\, s_2\, s_3 \ldots$ in $\mathcal{T}$ s.t.
$s_i \not\models a$ for infinitely many $i \geq 0$

iff there exists a reachable state s with $s \not\models a$
and a cycle $s \ldots s$

Persistence checking

given: finite transition system $\mathcal{T}$ over AP
persistence condition $a \in AP$

question: does $\mathcal{T} \models$ "eventually forever a" hold ?

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

iff there is a path $s_0\, s_1\, s_2\, s_3 \ldots$ in $\mathcal{T}$ s.t.
$s_i \not\models a$ for infinitely many $i \geq 0$

iff there exists a reachable state s with $s \not\models a$
and a cycle $s \ldots s$

iff there exists a non-trivial reachable SCC C
with $C \cap \{s \in S : s \not\models a\} \neq \varnothing$

Persistence checking

given: finite transition system $\mathcal{T}$ over AP
persistence condition $a \in AP$

question: does $\mathcal{T} \models$ "eventually forever a" hold ?

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

iff there exists a reachable state s with $s \not\models a$
and a cycle $s \ldots s$

iff there exists a non-trivial reachable **SCC** C
with $C \cap \{s \in S : s \not\models a\} \neq \emptyset$

SCC: strongly connected component, i.e., maximal set of states that are reachable from each other

Persistence checking

given: finite transition system $\mathcal{T}$ over AP
persistence condition $a \in AP$

question: does $\mathcal{T} \models$ "eventually forever a" hold ?

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

iff there exists a reachable state s with $s \not\models a$
and a cycle $s \ldots s$

iff there exists a non-trivial reachable **SCC** C
with $C \cap \{s \in S : s \not\models a\} \neq \emptyset$

A SCC is called non-trivial if it has at least one edge.
"either 1 state with a self-loop or 2 or more states"

Persistence checking

given: finite transition system $\mathcal{T}$ over AP
persistence condition $a \in AP$

question: does $\mathcal{T} \models$ "eventually forever a" hold ?

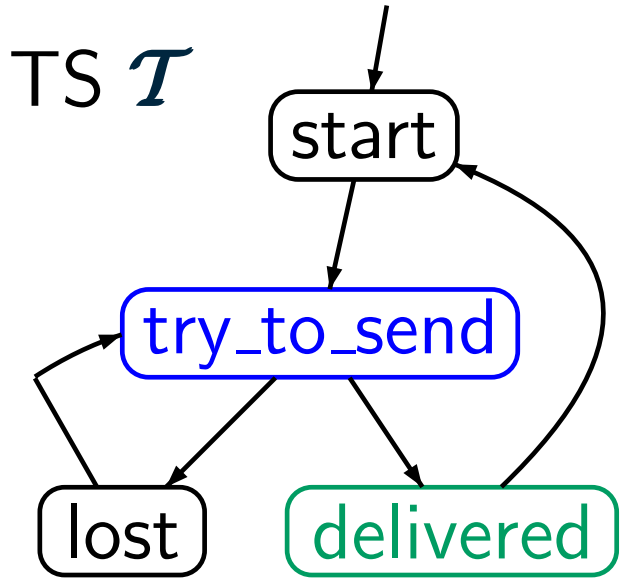
$\mathcal{T} \not\models$ "eventually forever a"

iff there exists a reachable state s with $s \not\models a$
and a cycle $s \ldots s$

iff there exists a non-trivial reachable **SCC** C
with $C \cap \{s \in S : s \not\models a\} \neq \emptyset$

method: calculate and analyze the **SCCs**

Example: ω-regular model checking

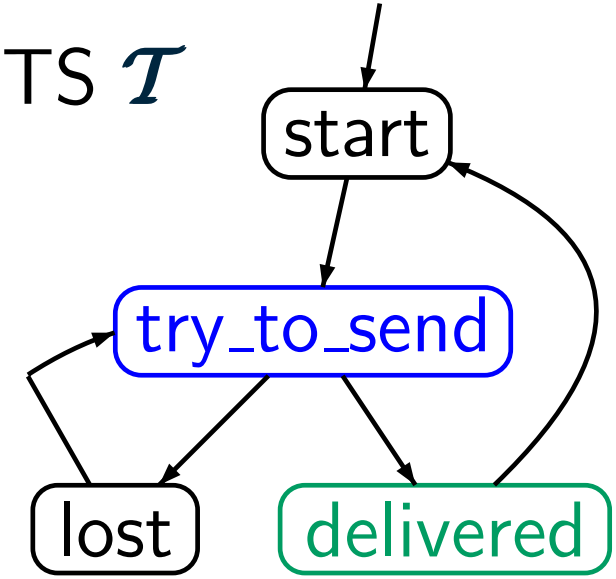


ω-regular LT property E:

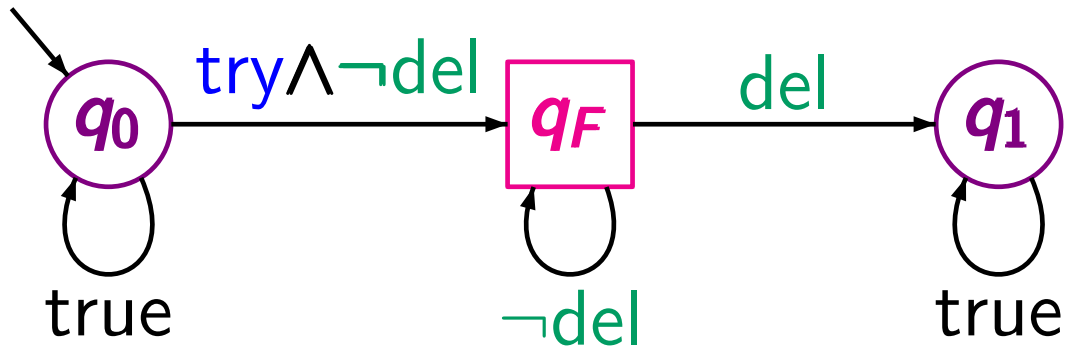
"each (repeatedly) sent message will eventually be delivered"

Example: ω-regular model checking

TS $\mathcal{T}$



NBA $\mathcal{A}$ for the bad behaviors

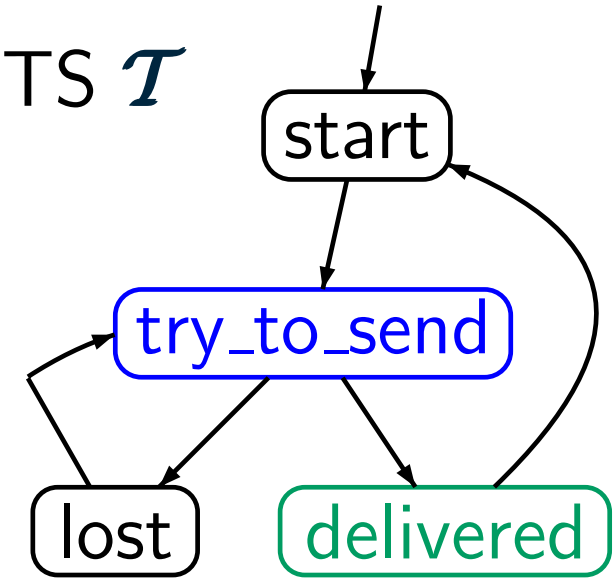


ω-regular LT property E:

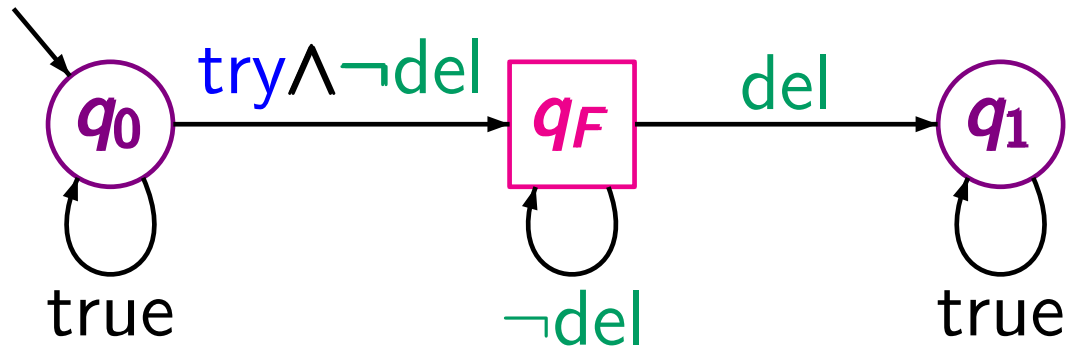
"each (repeatedly) sent message will eventually be delivered"

Example: ω-regular model checking

TS $\mathcal{T}$



NBA $\mathcal{A}$ for the bad behaviors

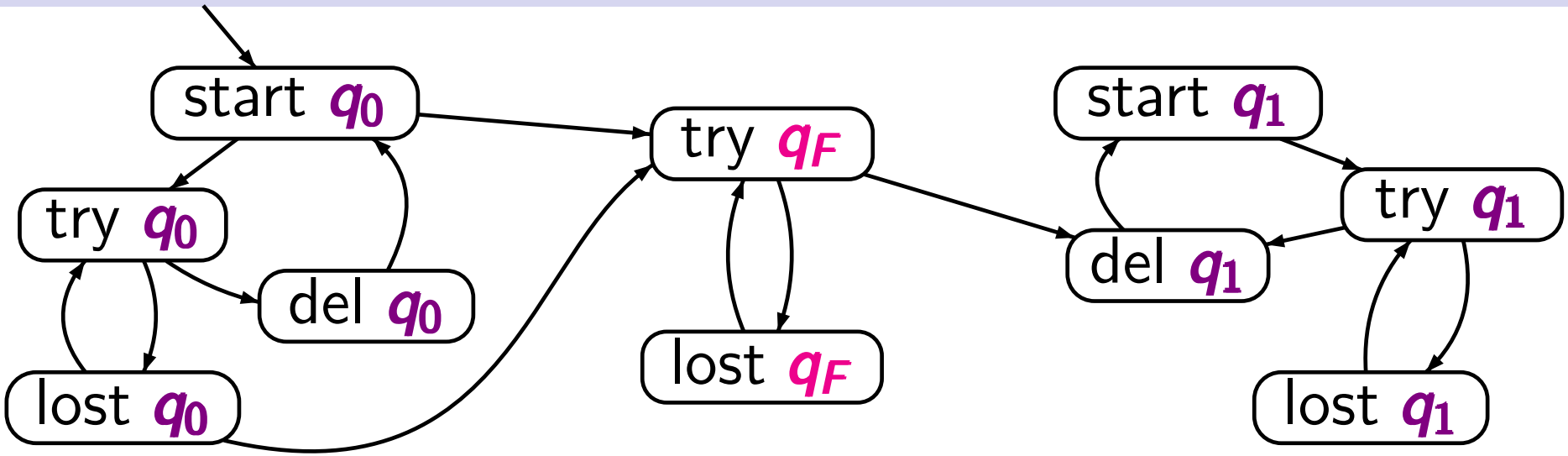


ω-regular LT property E:

"each (repeatedly) sent message will eventually be delivered"

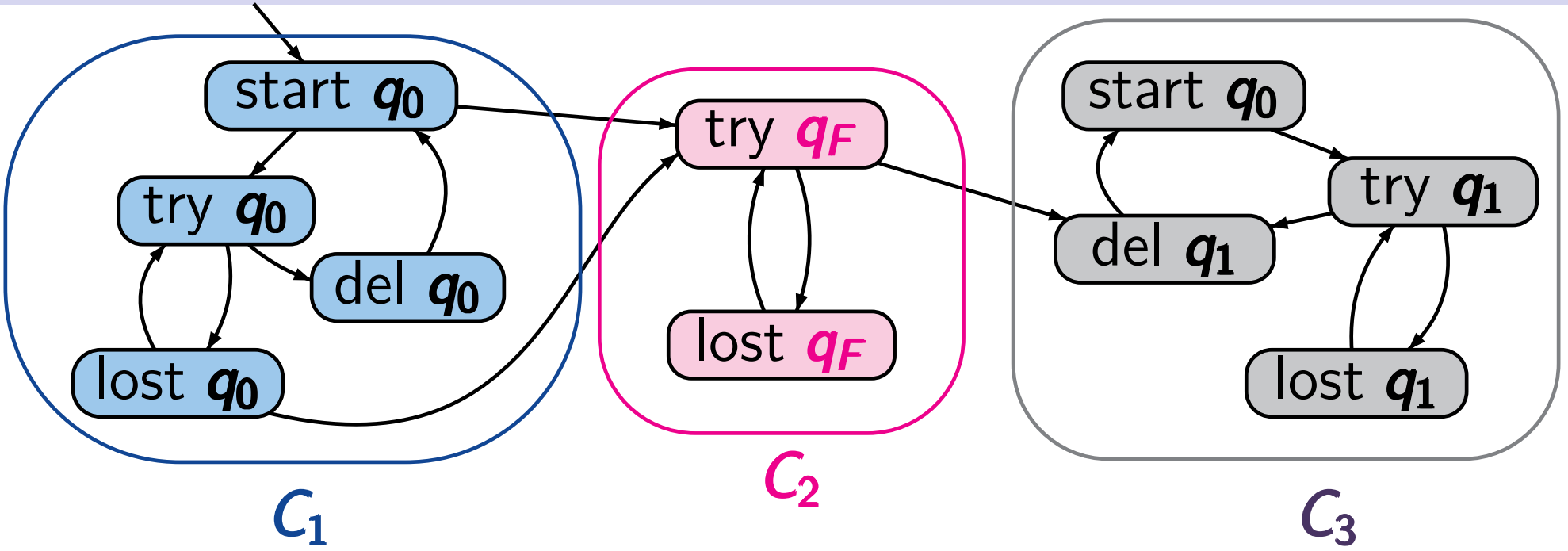
... analysis of the **SCCs** in product $\mathcal{T} \otimes \mathcal{A}$...

Example: persistence checking $\mathcal{T} \otimes \mathcal{A}$



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

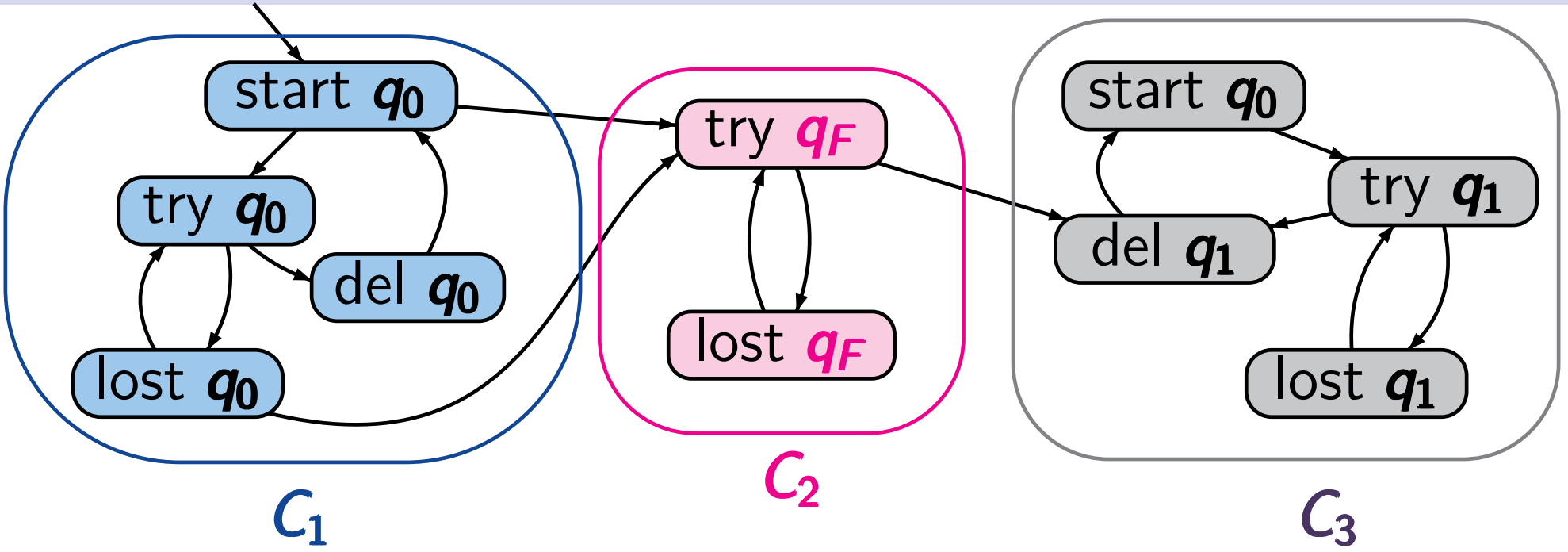
Example: persistence checking $\mathcal{T} \otimes \mathcal{A}$



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

3 reachable SCCs: C_1, C_2, C_3

Example: persistence checking $\mathcal{T} \otimes \mathcal{A}$

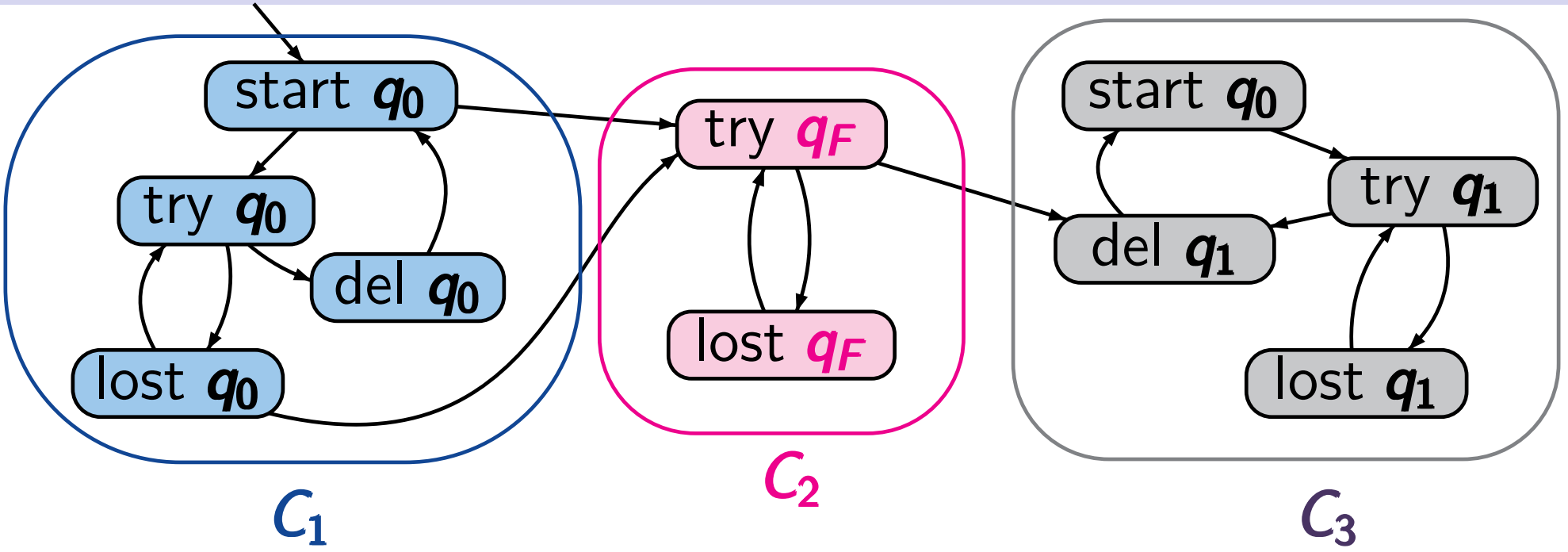


persistence property: "eventually forever $\neg q_F$"

3 reachable SCCs: C_1, C_2, C_3

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

Example: persistence checking $\mathcal{T} \otimes \mathcal{A}$



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 $\neg q_F$"

Persistence checking algorithms based on ...

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

iff there exists a reachable state s with $s \not\models a$ and a cycle $s \ldots s$

iff there exists a non-trivial reachable **SCC** C with $C \cap \{s \in S : s \not\models a\} \neq \emptyset$

Persistence checking algorithms based on ...

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

iff there exists a reachable state s with $s \not\models a$ and a cycle $s \ldots s$

iff there exists a non-trivial reachable **SCC** C with $C \cap \{s \in S : s \not\models a\} \neq \emptyset$

method 1: calculation and analysis of the SCCs

Persistence checking algorithms based on ...

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

iff there exists a reachable state s with $s \not\models a$ and a cycle $s \ldots s$

iff there exists a non-trivial reachable **SCC** C with $C \cap \{s \in S : s \not\models a\} \neq \emptyset$

method 1: calculation and analysis of the SCCs

- algorithm to compute the SCCs rely on an exploration of the full (reachable) state space
- not adequate for on-the-fly analysis

Persistence checking algorithms based on ...

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

iff there exists a reachable state s with $s \not\models a$ and a cycle $s \ldots s$

iff there exists a non-trivial reachable SCC C with $C \cap \{s \in S : s \not\models a\} \neq \varnothing$

method 1: calculation and analysis of the SCCs

- algorithm to compute the SCCs rely on an exploration of the full (reachable) state space
- not adequate for on-the-fly analysis

method 2: **DFS**-based search for **backward edges**

DFS-based cycle check

DFS-based cycle check

Let G be a finite directed graph and s a node in G.

The following statements are equivalent:

(1) G is cyclic

(2) The DFS in G finds some backward edge.

DFS-based cycle check

Let G be a finite directed graph and s a node in G.

The following statements are equivalent:

(1) G is cyclic

(2) The DFS in G finds some backward edge.

Cycle check in digraphs:

- perform by a DFS (with arbitrary starting node)
- check whether there is a backward edge

DFS-based cycle check

Let G be a finite directed graph and s a node in G.

The following statements are equivalent:

(1) G is cyclic

(2) The DFS in G finds some backward edge.

Cycle check in digraphs:

- perform by a DFS (with arbitrary starting node)
- check whether there is a backward edge

complexity: $\mathcal{O}(\text{size}(G))$

DFS-based cycle check

Let G be a finite directed graph and s a node in G.

The following statements are equivalent:

(1) s belongs to a cycle $s\, s_1\, s_2 \ldots s_k\, s$

(2) The DFS started with s finds a backward edge $s' \longrightarrow s$.

Cycle check for fixed node: "does s belong to a cycle?"

- perform by a DFS with starting node s
- check whether there is a backward edge $s' \longrightarrow s$

complexity: $\mathcal{O}(\text{size}(G))$

DFS-based persistence checking

given: finite TS $\mathcal{T}$, persistence condition a

question: does $\mathcal{T} \models$ "eventually forever a" hold?

DFS-based persistence checking

given: finite TS $\mathcal{T}$, persistence condition a

question: does $\mathcal{T} \models$ "eventually forever a" hold?

initially all states are unmarked

REPEAT

  choose an unmarked reachable state s with $s \not\models a$;
  mark s;
  IF *CYCLE_CHECK*(s) THEN
    return "no"
  FI

UNTIL all reachable states s with $s \not\models a$ are marked;
return "yes"

DFS-based persistence checking

given: finite TS $\mathcal{T}$, persistence condition a

question: does $\mathcal{T} \models$ "eventually forever a" hold?

initially all states are unmarked

1. DFS: visits all reachable states

```
REPEAT
   choose an unmarked reachable state s with s ⊭ a;
   mark s;
   IF CYCLE_CHECK(s) THEN
     return "no"
   FI
UNTIL all reachable states s with s ⊭ a are marked;
return "yes"
```

DFS-based persistence checking

given: finite TS $\mathcal{T}$, persistence condition a

question: does $\mathcal{T} \models$ "eventually forever a" hold?

initially all states are unmarked

REPEAT

1. DFS: visits all reachable states

choose an unmarked reachable state s with $s \not\models a$;

mark s;

IF *CYCLE_CHECK*(s) THEN

return "no"

FI

2. DFS: searches for a backward edge $s' \rightarrow s$

UNTIL all reachable states s with $s \not\models a$ are marked;

return "yes"

Persistence checking ⟵ Nested DFS

given: finite TS $\mathcal{T}$, persistence condition a

question: does $\mathcal{T} \models$ "eventually forever a" hold?

initially all states are unmarked

REPEAT

1. DFS: visits all reachable states

choose an unmarked reachable state s with $s \not\models a$;

mark s;

IF *CYCLE_CHECK*(s) THEN

return "no"

FI

2. DFS: searches for a backward edge $s' \rightarrow s$

UNTIL all reachable states s with $s \not\models a$ are marked;

return "yes"

Time complexity of nested DFS

Time complexity of nested DFS

1. DFS: visits all reachable states

```
REPEAT
    choose an unmarked reachable state s with s ⊭ a;
    mark s;
    IF CYCLE_CHECK(s) THEN
      return "no"
    FI
UNTIL all reachable states s with s ⊭ a are marked;
return "yes"
```

2. DFS: searches for a backward edge $s' \longrightarrow s$

worst case: $\Theta(|S| \cdot (|S| + \#\mathbf{edges}))$ naïve approach

Time complexity of nested DFS

REPEAT

1. DFS: visits all reachable states

  choose an unmarked reachable state s with $s \not\models a$;
  mark s;
  IF *CYCLE_CHECK*(s) THEN
    return "no"
  FI

2. DFS: searches for a backward edge $s' \rightarrow s$

UNTIL all reachable states s with $s \not\models a$ are marked;
return "yes"

worst case: $\Theta(|S| \cdot (\underbrace{|S| + \#\mathbf{edges}}))$ naïve approach

cost of *CYCLE_CHECK*(s) caused by each state $s \not\models a$

Time complexity of nested DFS

REPEAT

1. DFS: visits all reachable states

  choose an unmarked reachable state s with $s \not\models a$;
  mark s;
  IF *CYCLE_CHECK*(s) THEN
    return "no"
  FI

2. DFS: searches for a backward edge $s' \rightarrow s$

UNTIL all reachable states s with $s \not\models a$ are marked;
return "yes"

worst case: $\Theta(|S| \cdot (|S| + \#\mathbf{edges}))$ naïve approach

$\Theta(|S|)$ states with $s \not\models a$

cost of *CYCLE_CHECK*(s) caused by each state $s \not\models a$

Time complexity of nested DFS

1. DFS: visits all reachable states

REPEAT

choose an unmarked reachable state s with $s \not\models a$;
mark s;
IF *CYCLE_CHECK*(s) THEN
return "no"
FI

2. DFS: searches for a backward edge $s' \rightarrow s$

UNTIL all reachable states s with $s \not\models a$ are marked;
return "yes"

complexity: $\Theta(|S| \cdot (|S| + \#\mathbf{edges}))$ "tricky" variant

Nested DFS

- serves to check whether $\mathcal{T} \models$ "eventually forever a"
- relies on two DFS running in an interleaved way

Nested DFS

- serves to check whether $\mathcal{T} \models$ "eventually forever a"
- relies on two DFS running in an interleaved way

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(s) for states s with $s \not\models a$

checks whether s belongs to a cycle

Nested DFS

- serves to check whether $\mathcal{T} \models$ "eventually forever a"
- relies on two DFS running in an interleaved way

1. DFS: visits all reachable states

2. DFS: *CYCLE_CHECK*(s) for states s with $s \not\models a$

checks whether s belongs to a cycle

- by searching a backward edge $s' \rightarrow s$

Nested DFS

- serves to check whether $\mathcal{T} \models$ "eventually forever a"
- relies on two DFS running in an interleaved way

1. DFS: visits all reachable states

2. DFS: *CYCLE_CHECK*(s) for states s with $s \not\models a$

checks whether s belongs to a cycle

- by searching a backward edge $s' \to s$
- ignores states that have been visited in previous calls of *CYCLE_CHECK*

Nested DFS

- serves to check whether $\mathcal{T} \models$ "eventually forever a"
- relies on two DFS running in an interleaved way

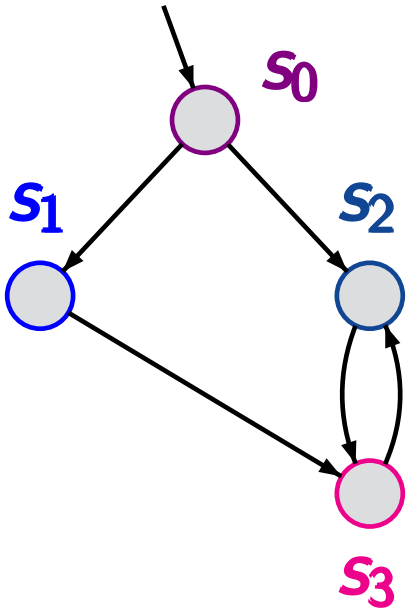
1. DFS: visits all reachable states

2. DFS: *CYCLE_CHECK*(s) for states s with $s \not\models a$

checks whether s belongs to a cycle

- by searching a backward edge $s' \to s$
- ignores states that have been visited in previous calls of *CYCLE_CHECK*
- uses a global visiting set V of states that have been visited so far in the **2. DFS**

Example: nested DFS

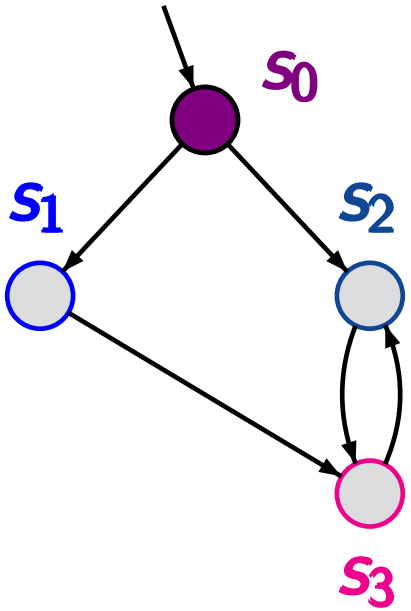


$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



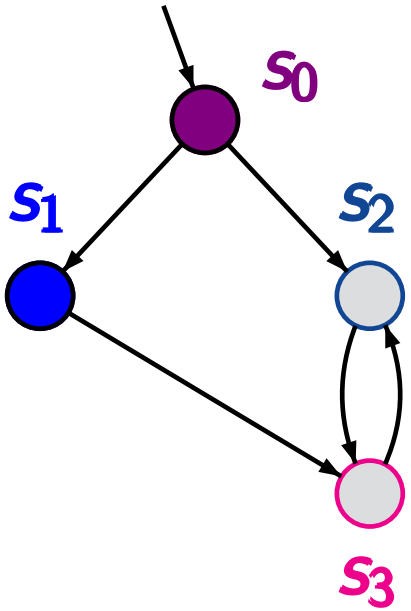
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

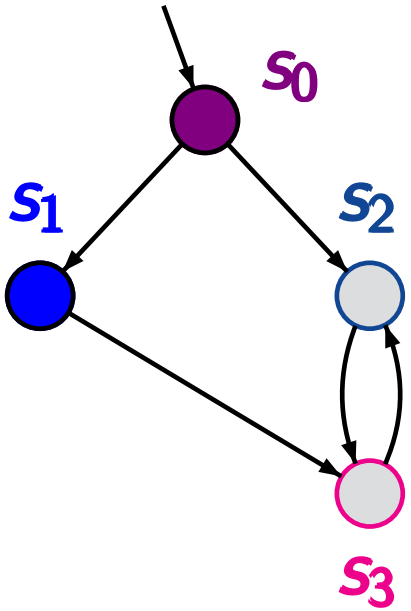
$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

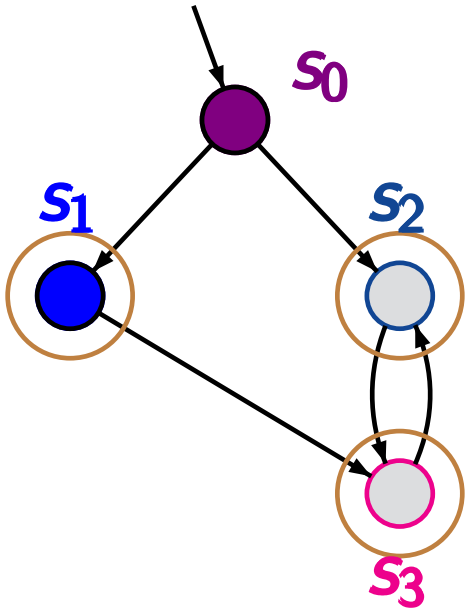
$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

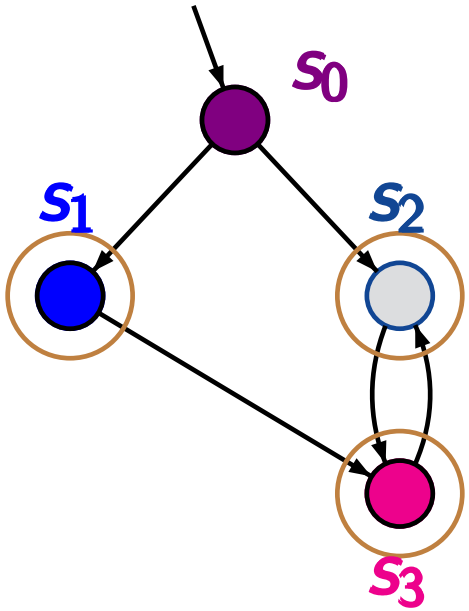
$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

$V = \{s_1, s_2, s_3\}$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

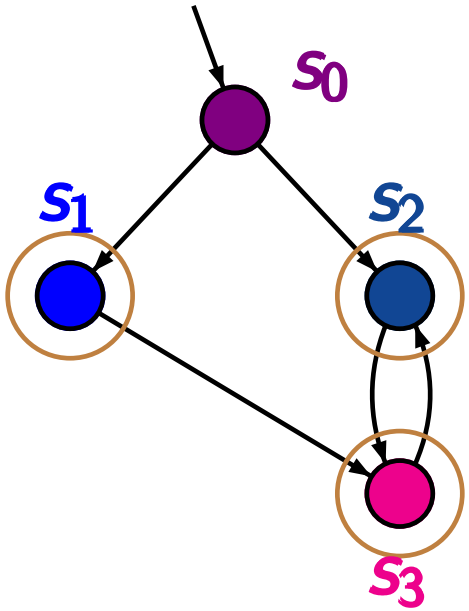
$CYCLE\_CHECK(s_1)$

$V = \{s_1, s_2, s_3\}$

$DFS(s_3)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

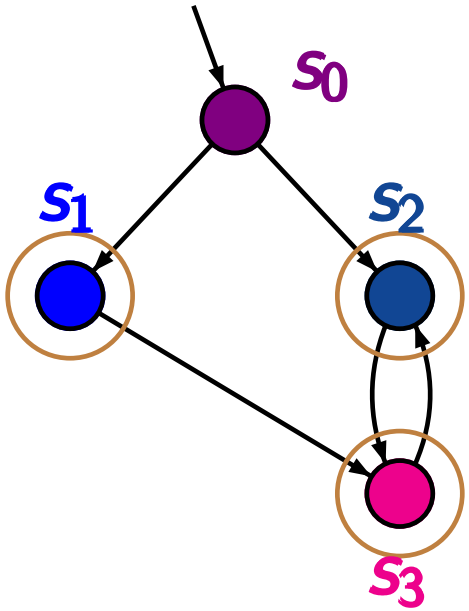
$V = \{s_1, s_2, s_3\}$

$DFS(s_3)$

$DFS(s_2)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

$V = \{s_1, s_2, s_3\}$

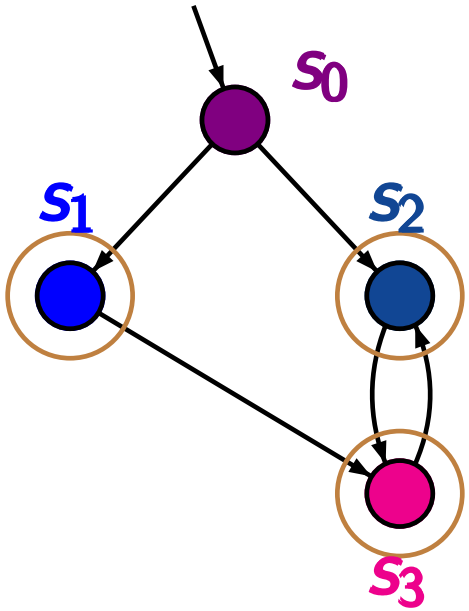
$DFS(s_3)$

$DFS(s_2)$

$CYCLE\_CHECK(s_2)$

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

Example: nested DFS ⟵ fails



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

$V = \{s_1, s_2, s_3\}$

$DFS(s_3)$

$DFS(s_2)$

$CYCLE\_CHECK(s_2)$

↑

returns wrong answer "**yes**"

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

Nested DFS

- serves to check whether $\mathcal{T} \models$ "eventually forever a"
- relies on two DFSs running in an interleaved way

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(s) for states s with $s \not\models a$

Nested DFS

- serves to check whether $\mathcal{T} \models$ "eventually forever a"
- relies on two DFSs running in an interleaved way

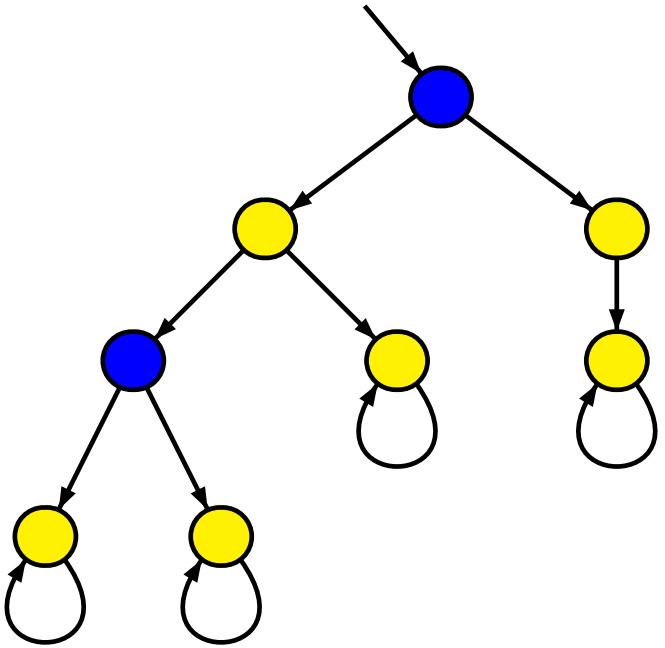
1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(s) for states s with $s \not\models a$
   - checks whether s belongs to a cycle by searching a backward edge $s' \rightarrow s$
   - ignores states that have been visited in previous calls of *CYCLE_CHECK* by using a global visiting set V for the **2. DFS**

Nested DFS

- serves to check whether $\mathcal{T} \models$ "eventually forever a"
- relies on two DFSs running in an interleaved way

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(s) for states s with $s \not\models a$
   - checks whether s belongs to a cycle by searching a backward edge $s' \rightarrow s$
   - ignores states that have been visited in previous calls of *CYCLE_CHECK* by using a global visiting set V for the **2. DFS**
   - is called for state s after s is **fully expanded** in the **1. DFS**

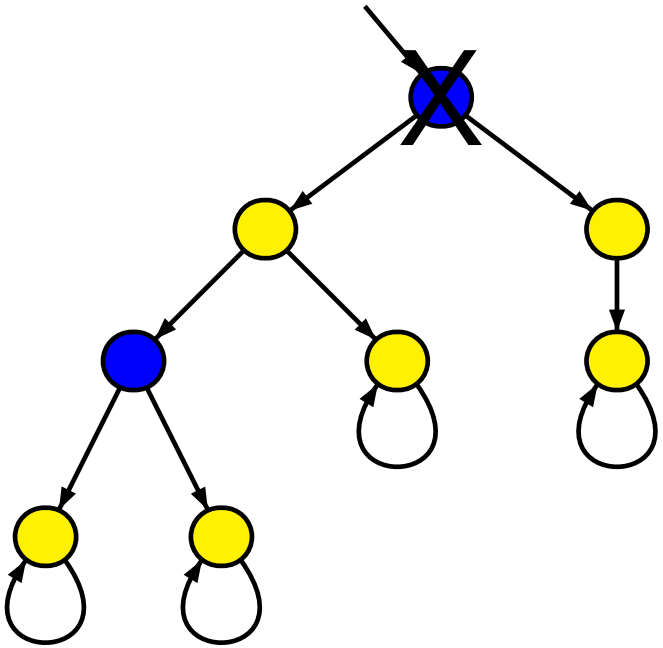
Nested DFS



$\mathcal{T} \models$ "eventually forever $\neg$*blue*"

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(***s***) for $s \models$ ***blue***
   checks whether ***s*** belongs to a cycle
   by searching a backward-edge $s' \rightarrow s$

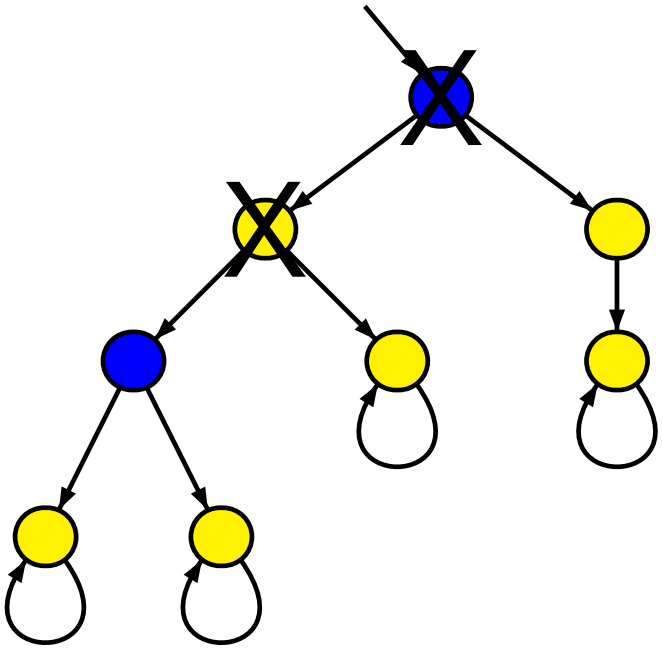
Nested DFS



$\mathcal{T} \models$ "eventually forever $\neg$*blue*"

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(*s*) for $s \models$ *blue*
   checks whether *s* belongs to a cycle
   by searching a backward-edge $s' \longrightarrow s$

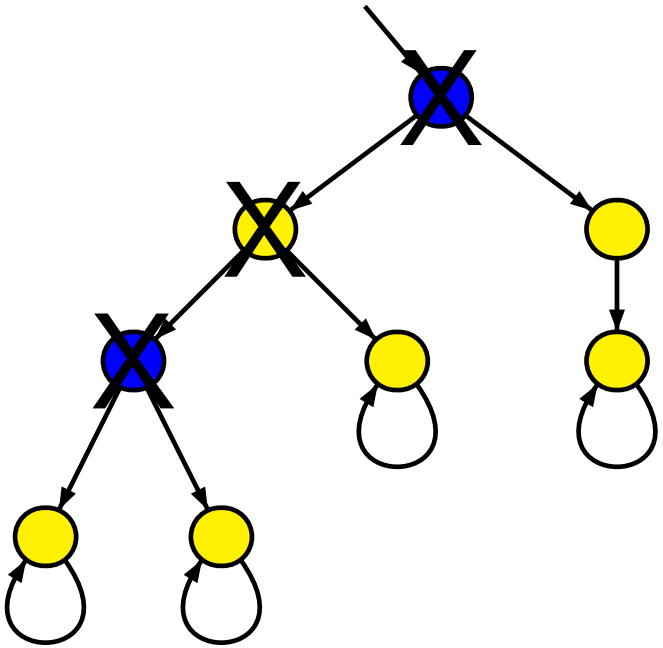
Nested DFS



$\mathcal{T} \models$ "eventually forever $\neg$*blue*"

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(*s*) for $s \models$ *blue*
   checks whether *s* belongs to a cycle
   by searching a backward-edge $s' \rightarrow s$

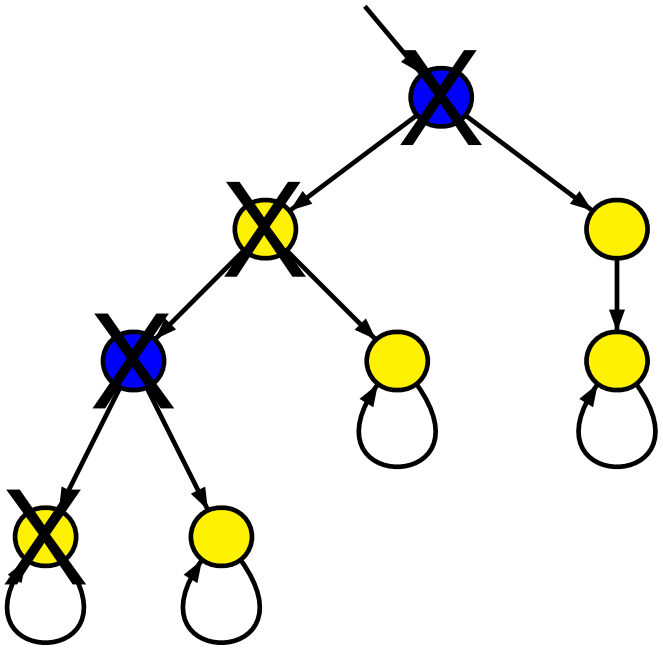
Nested DFS



$\mathcal{T} \models$ "eventually forever $\neg$*blue*"

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(*s*) for $s \models$ *blue*
   checks whether *s* belongs to a cycle
   by searching a backward-edge $s' \longrightarrow s$

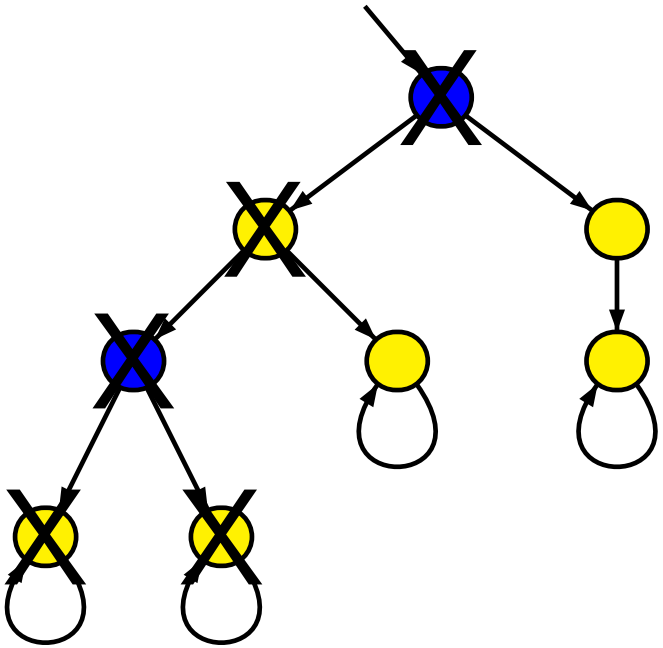
Nested DFS



$\mathcal{T} \models$ "eventually forever $\neg$*blue*"

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(*s*) for $s \models$ *blue*
   checks whether *s* belongs to a cycle
   by searching a backward-edge $s' \rightarrow s$

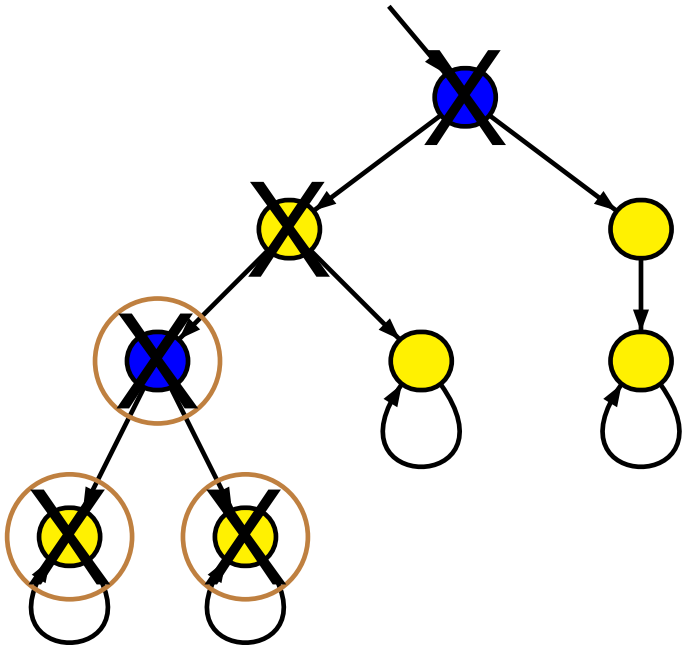
Nested DFS



$\mathcal{T} \models$ "eventually forever $\neg$*blue*"

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(*s*) for $s \models$ *blue*
   checks whether *s* belongs to a cycle
   by searching a backward-edge $s' \longrightarrow s$

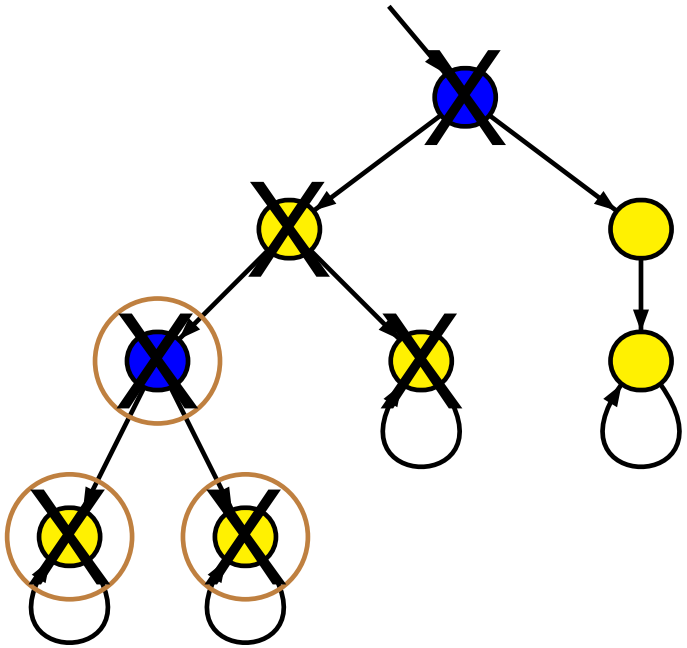
Nested DFS



$\mathcal{T} \models$ "eventually forever $\neg$*blue*"

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(*s*) for $s \models$ *blue*
   checks whether *s* belongs to a cycle
   by searching a backward-edge $s' \rightarrow s$

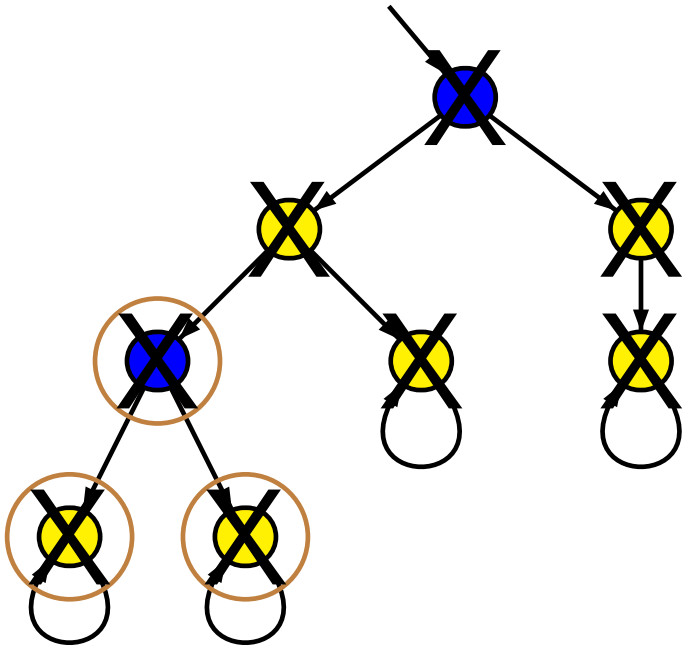
Nested DFS



$\mathcal{T} \models$ "eventually forever $\neg$*blue*"

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(*s*) for $s \models$ *blue*
   checks whether *s* belongs to a cycle
   by searching a backward-edge $s' \longrightarrow s$

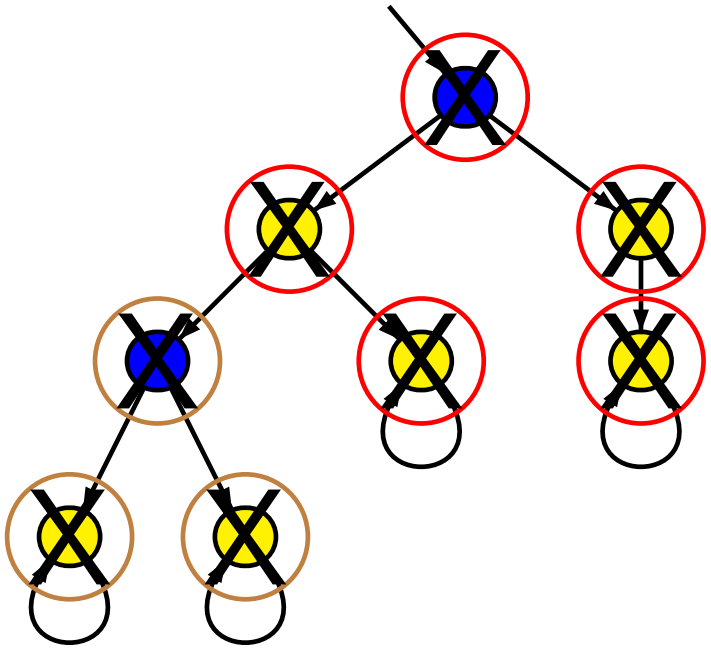
Nested DFS



$\mathcal{T} \models$ "eventually forever $\neg$*blue*"

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(*s*) for $s \models$ *blue*
   checks whether *s* belongs to a cycle
   by searching a backward-edge $s' \longrightarrow s$

Nested DFS



$\mathcal{T} \models$ "eventually forever $\neg$*blue*"

1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(*s*) for $s \models$ *blue*
   checks whether *s* belongs to a cycle
   by searching a backward-edge $s' \rightarrow s$

Nested DFS (pseudo code)

$U := \emptyset$;

FOR ALL $s_0 \in S_0$ DO $DFS(s_0)$ OD ;

Nested DFS (pseudo code)

$U := \emptyset$; ⟵ visiting set of 1. DFS

FOR ALL $s_0 \in S_0$ DO $DFS(s_0)$ OD ;

Nested DFS (pseudo code)

$U := \emptyset$; ⟵ visiting set of 1. DFS

FOR ALL $s_0 \in S_0$ DO $DFS(s_0)$ OD ;

IF $s \notin U$ THEN
    insert s in U;

pseudo code for $DFS(s)$

Nested DFS (pseudo code)

$U := \emptyset$; ⟵ visiting set of 1. DFS

FOR ALL $s_0 \in S_0$ DO $DFS(s_0)$ OD ;

IF $s \notin U$ THEN
  insert s in U;
  FOR ALL $s' \in Post(s)$ DO $DFS(s')$ OD ;

pseudo code for $DFS(s)$

Nested DFS (pseudo code)

$U := \emptyset$; ⟵ visiting set of 1. DFS

FOR ALL $s_0 \in S_0$ DO $DFS(s_0)$ OD ;

pseudo code for $DFS(s)$

IF $s \notin U$ THEN
  insert s in U;
  FOR ALL $s' \in Post(s)$ DO $DFS(s')$ OD ;
  IF $s \not\models a$ THEN
    IF $CYCLE\_CHECK(s)$
  FI

Nested DFS (pseudo code)

$U := \emptyset$; ⟵ visiting set of 1. DFS

$V := \emptyset$ ⟵ global visiting set of 2. DFS

```
FOR ALL s0 ∈ S0 DO DFS(s0) OD ;
```

pseudo code for $DFS(s)$

```
IF s ∉ U THEN
    insert s in U;
    FOR ALL s' ∈ Post(s) DO DFS(s') OD ;
    IF s ⊭ a THEN
        IF CYCLE_CHECK(s)
    FI
```

Nested DFS (pseudo code)

$U := \emptyset$; ⟵ visiting set of 1. DFS

$V := \emptyset$ ⟵ global visiting set of 2. DFS

FOR ALL $s_0 \in S_0$ DO $DFS(s_0)$ OD ;

pseudo code for $DFS(s)$

IF $s \notin U$ THEN
    insert s in U;
    FOR ALL $s' \in Post(s)$ DO $DFS(s')$ OD ;
    IF $s \not\models a$ THEN
        IF $CYCLE\_CHECK(s)$ THEN return "**no**" FI
    FI

Nested DFS (pseudo code)

$U := \emptyset$; ⟵ visiting set of 1. DFS

$V := \emptyset$ ⟵ global visiting set of 2. DFS

```
FOR ALL s0 ∈ S0 DO DFS(s0) OD ;
```

pseudo code for $DFS(s)$

```
IF s ∉ U THEN
    insert s in U;
    FOR ALL s' ∈ Post(s) DO DFS(s') OD ;
    IF s ⊭ a THEN
        IF CYCLE_CHECK(s) THEN return "no" FI
    FI
```

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

Nested DFS (pseudo code)

$U := \emptyset$; ⟵ visiting set of 1. DFS

$V := \emptyset$ ⟵ global visiting set of 2. DFS

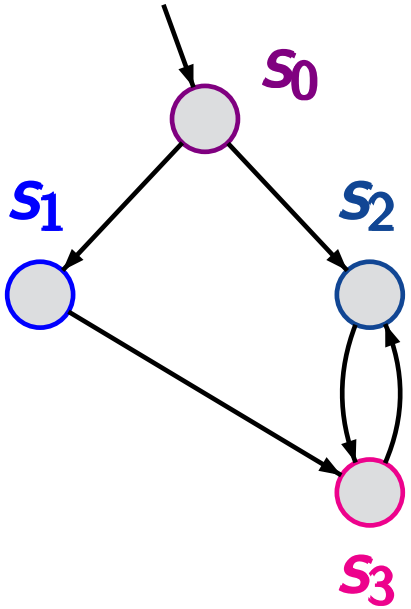
FOR ALL $s_0 \in S_0$ DO $DFS(s_0)$ OD ;

return "**yes**" ⟵ $\mathcal{T} \models$ "eventually forever a"

pseudo code for $DFS(s)$

```
IF s ∉ U THEN
    insert s in U;
    FOR ALL s' ∈ Post(s) DO DFS(s') OD ;
    IF s ⊭ a THEN
        IF CYCLE_CHECK(s) THEN return "no" FI
    FI
FI
```

Example: nested DFS

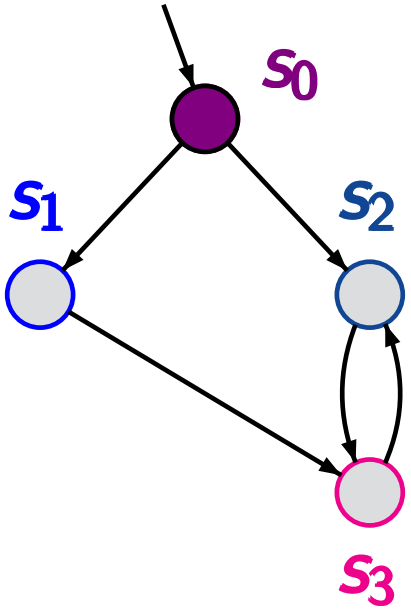


$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$\mathcal{T} \not\models$ “eventually forever a”

Example: nested DFS



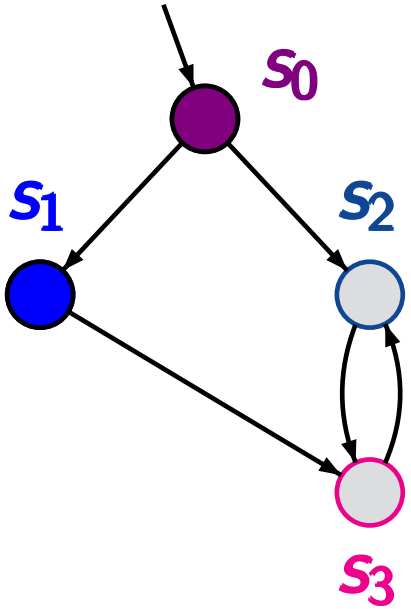
$DFS(s_0)$

$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



$DFS(s_0)$

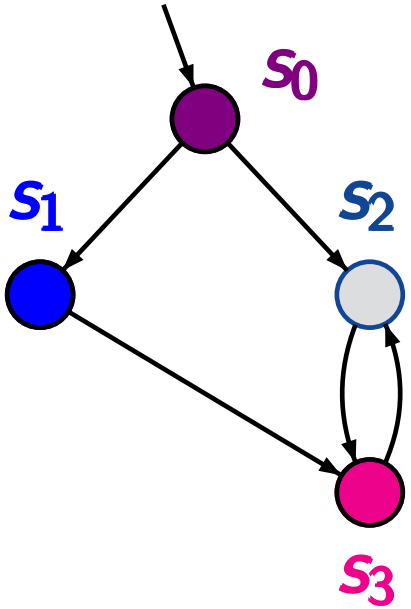
$DFS(s_1)$

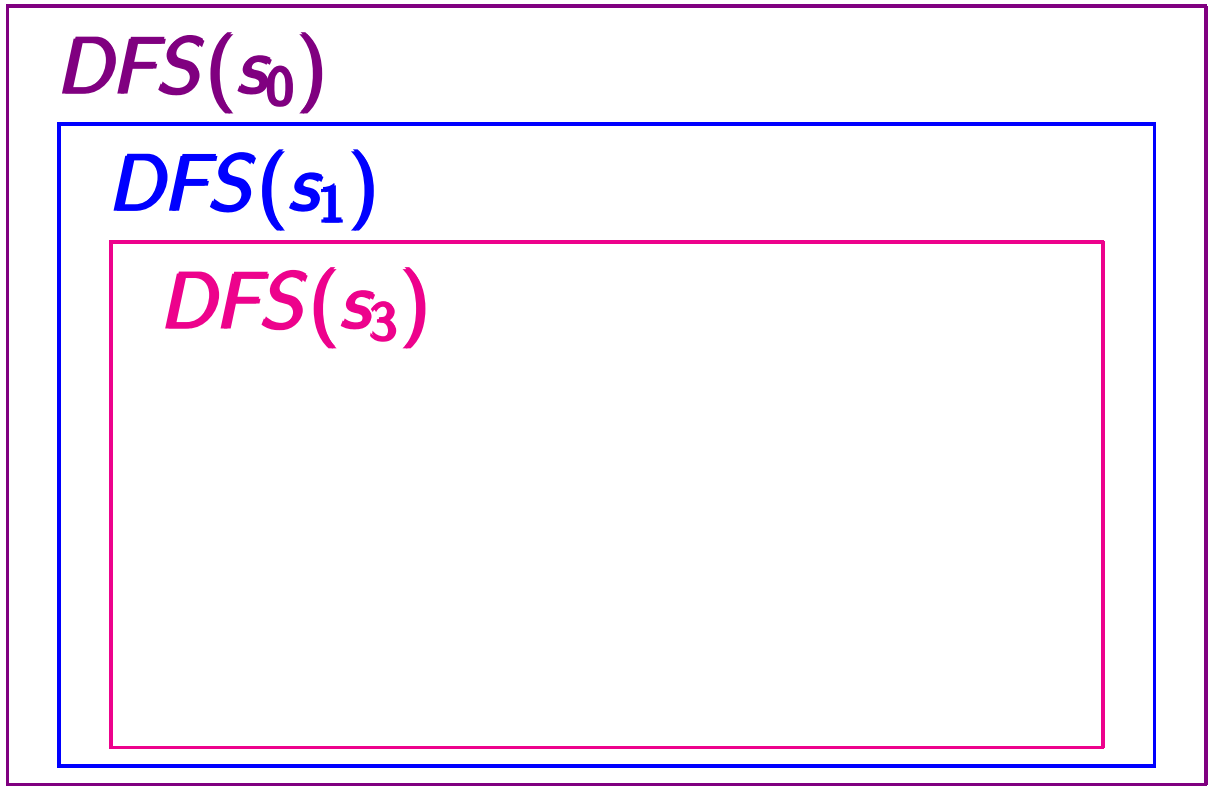
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



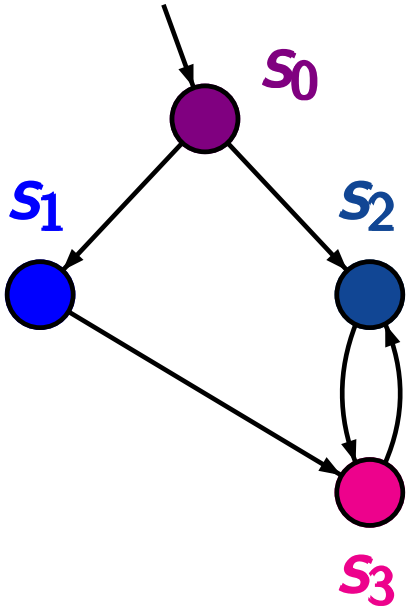


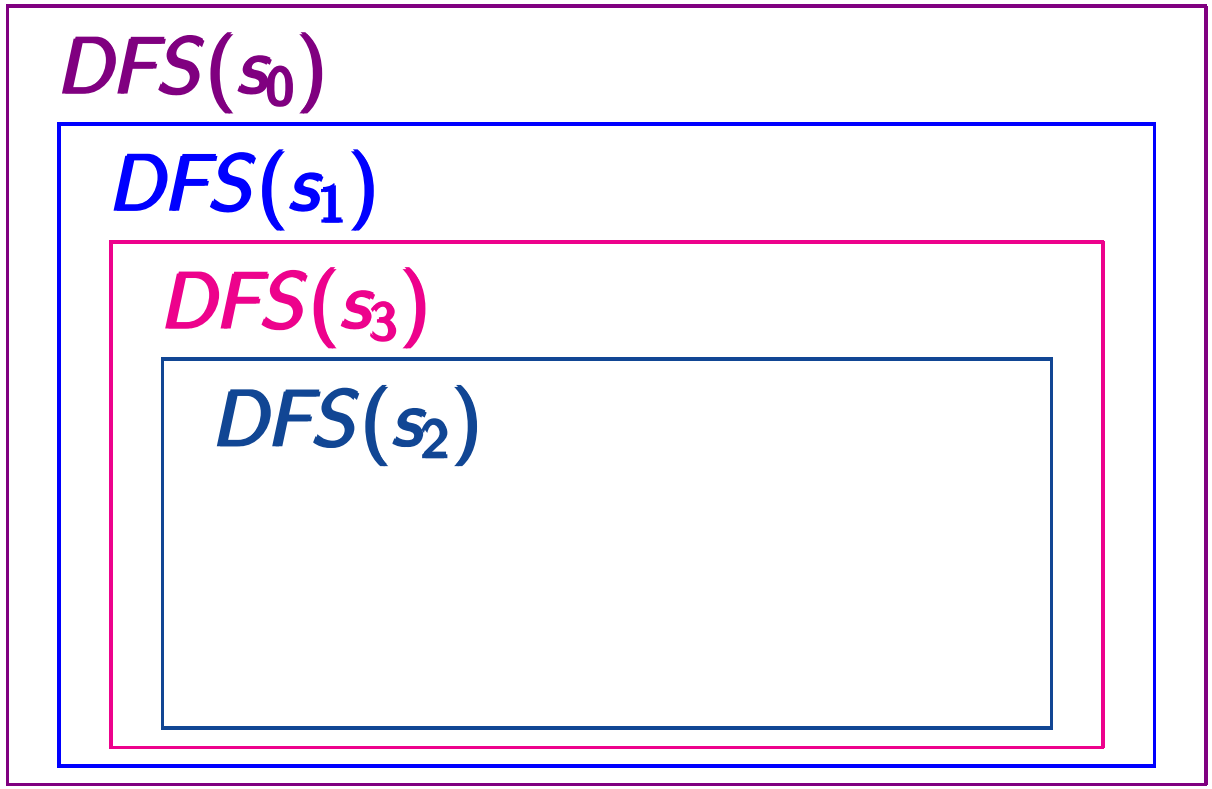
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



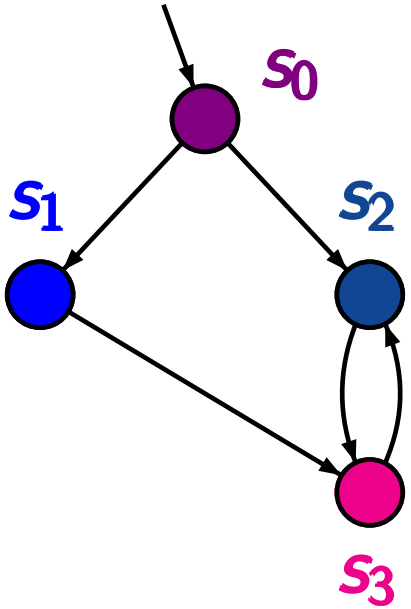


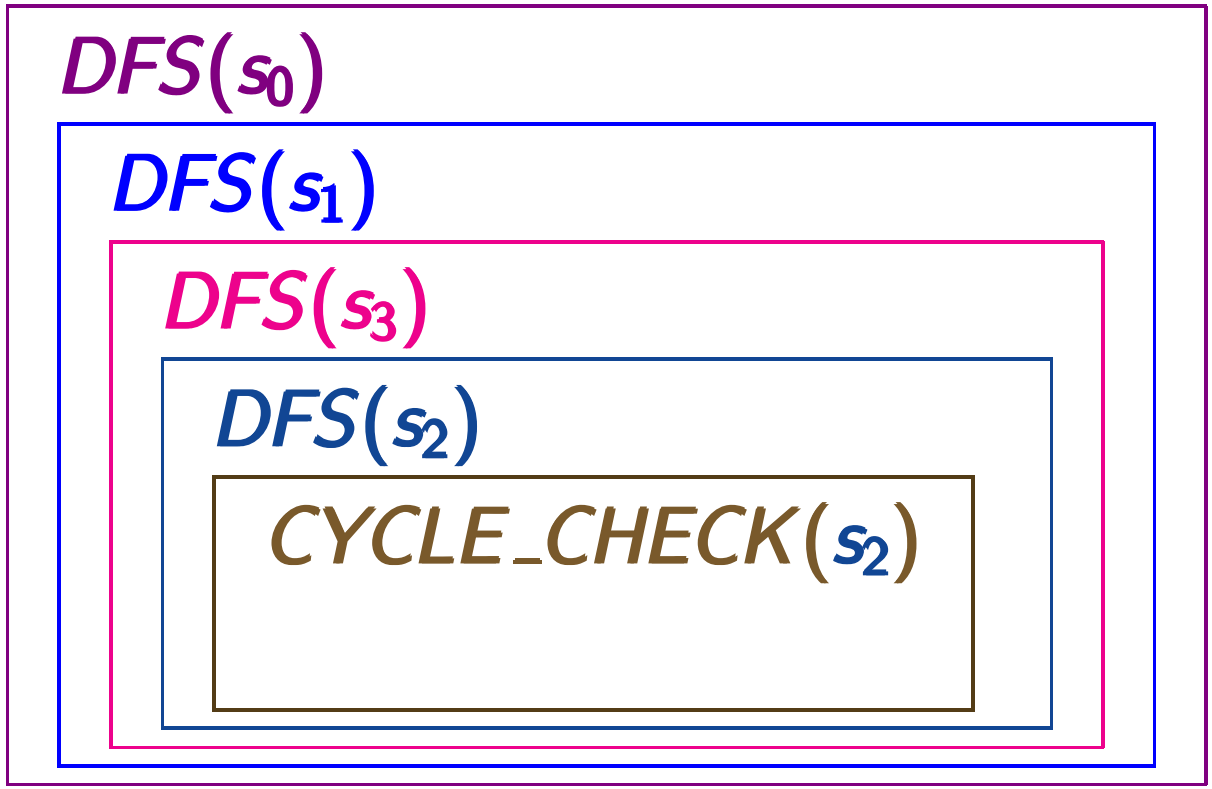
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



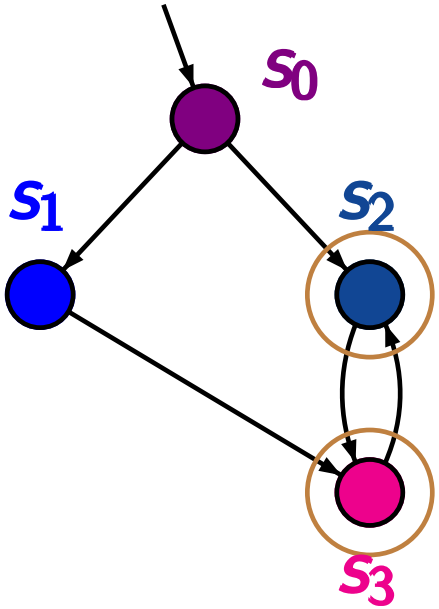


$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

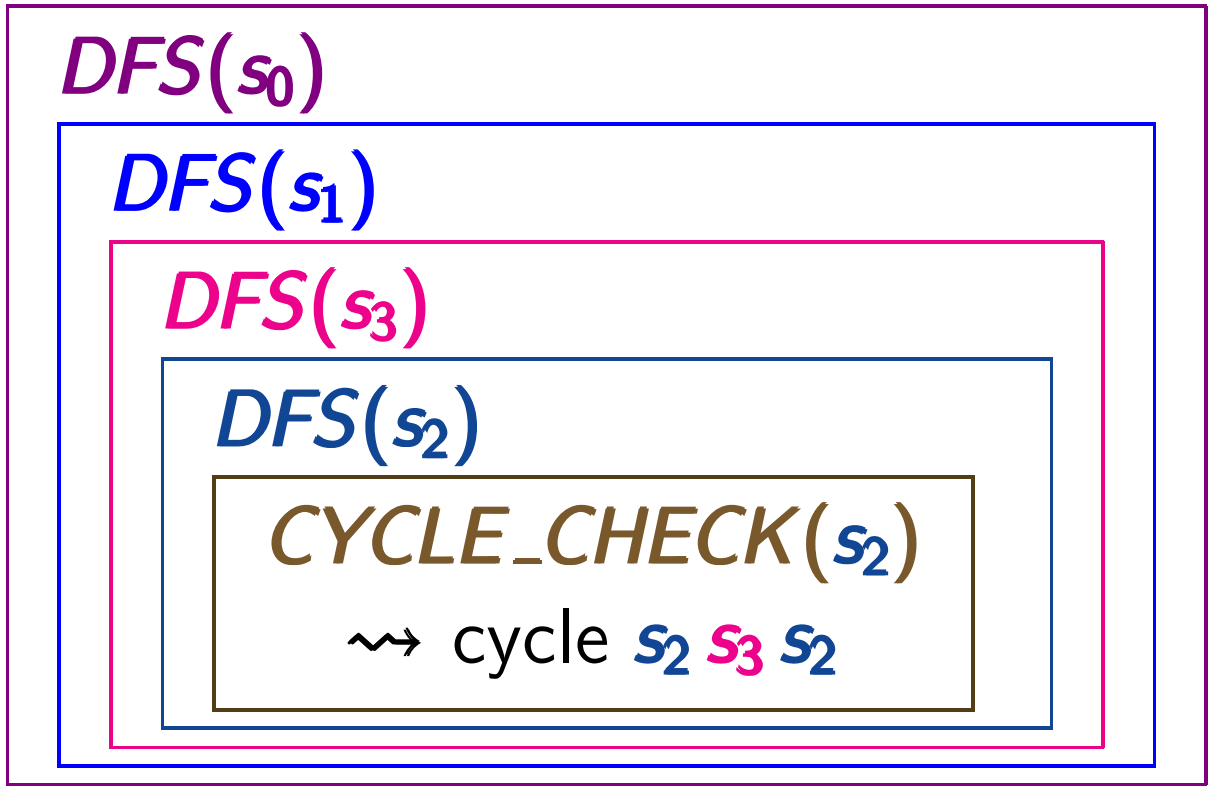
$\mathcal{T} \not\models$ "eventually forever a"

Example: nested DFS



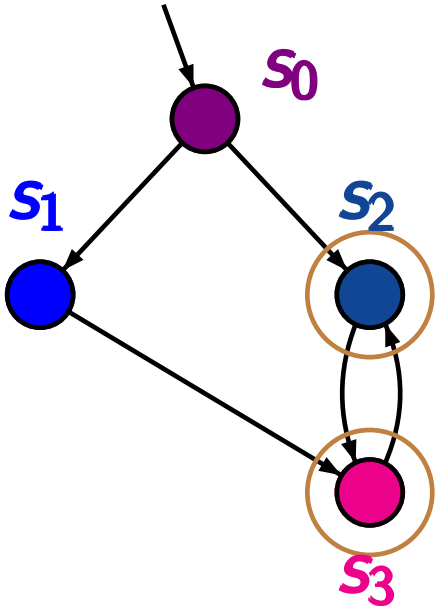
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$



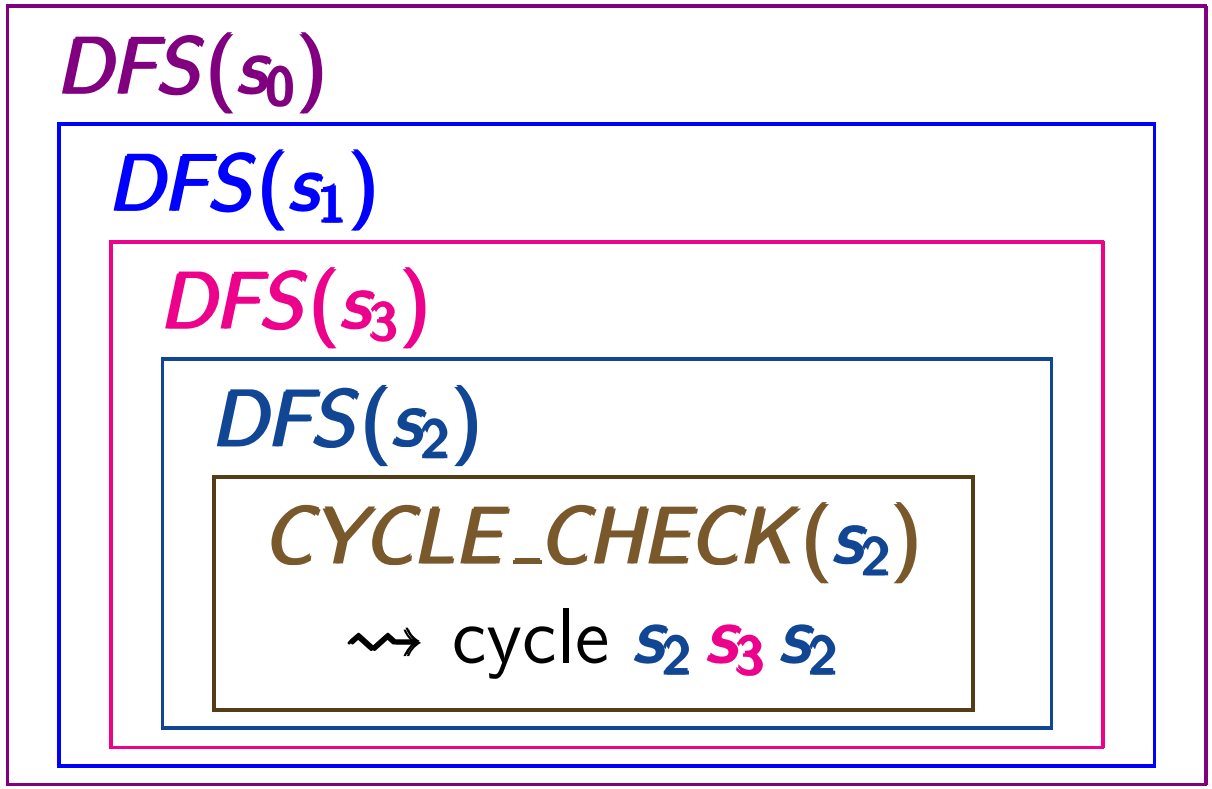
$\mathcal{T} \not\models$ "eventually forever a"

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$



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

↑
returns correct
answer "**no**"

Nested DFS with counterexample generation LTLMC3.2-34

Nested DFS with counterexample generation

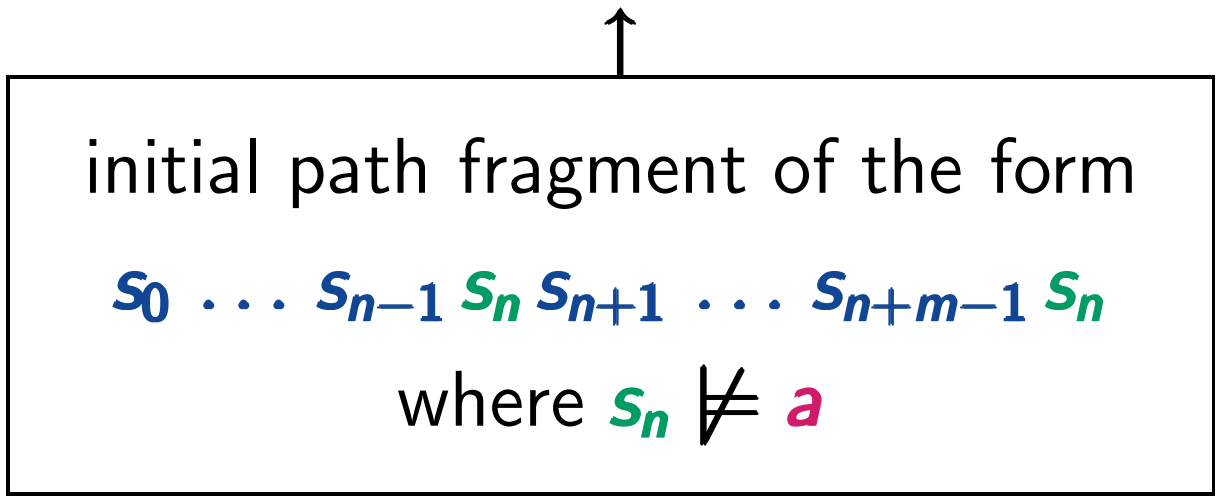
input: finite TS $\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$
persistence condition $a \in AP$

output: "yes" if $\mathcal{T} \models$ "eventually forever a"
"no" + counterexample otherwise

Nested DFS with counterexample generation

input: finite TS $\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$
persistence condition $a \in AP$

output: "yes" if $\mathcal{T} \models$ "eventually forever a"
"no" + counterexample otherwise



Nested DFS with counterexample generation

input: finite TS $\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$
persistence condition $a \in AP$

output: "yes" if $\mathcal{T} \models$ "eventually forever a"
"no" + counterexample otherwise

initial path fragment of the form

$$s_0 \ldots s_{n-1} s_n s_{n+1} \ldots s_{n+m-1} s_n$$

where $s_n \not\models a$

... iterative formulation with 2 stacks ...

$U := \varnothing$; $\pi := \varnothing$;

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

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

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

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

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

WHILE $S_0 \not\subseteq U$ DO

OD

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

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

WHILE $S_0 \not\subseteq U$ DO

choose $s_0 \in S_0 \setminus U$; insert s_0 in U;

OD

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

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

WHILE $S_0 \not\subseteq U$ DO

choose $s_0 \in S_0 \setminus U$; insert s_0 in U; $Push(\pi, s_0)$;

OD

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

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

WHILE $S_0 \not\subseteq U$ DO

choose $s_0 \in S_0 \setminus U$; insert s_0 in U; $Push(\pi, s_0)$;

WHILE $\pi \neq \emptyset$ DO

$s := Top(\pi)$;

OD OD

$U := \emptyset$; $\pi := \emptyset$; ← 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
```

```
OD OD   FI
```

$U := \emptyset$; $\pi := \emptyset$; ← 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')

    FI
OD OD
```

$U := \emptyset$; $\pi := \emptyset$; ← 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);

    FI
OD OD
```

$U := \emptyset$; $\pi := \emptyset$; ← 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 ⊭ a and CYCLE_CHECK(s)
                THEN return "no"                FI
    FI
OD OD
```

$U := \emptyset$; $\pi := \emptyset$; ← 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 ⊭ a and CYCLE_CHECK(s)
                 THEN return "no" + reverse(π, ξ) FI
    FI
OD OD
```

$U := \emptyset$; $\pi := \emptyset$; ← 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 ⊭ a and CYCLE_CHECK(s)
                 THEN return "no" + reverse(π, ξ) FI
        FI
OD OD
return "yes"
```

Algorithm *CYCLE_CHECK*(*s*)

- is called for $s \not\models a$
- checks whether s belongs to a cycle
- uses global visiting set V and stack ξ

Algorithm *CYCLE_CHECK*(*s*)

Push(ξ, *s*); insert *s* in *V*;

Algorithm *CYCLE_CHECK*(*s*)

Push(ξ, *s*); insert *s* in *V*;

WHILE $\xi \neq \varnothing$ DO

Algorithm *CYCLE_CHECK*(s)

Push(ξ, s); insert s in V;

WHILE $\xi \neq \varnothing$ DO

    $s' :=$ *Top*(ξ);

    IF $s \in$ *Post*(s')

Algorithm *CYCLE_CHECK*(s)

Push(ξ, s); insert s in V;

WHILE $\xi \neq \varnothing$ DO

  $s' :=$ *Top*(ξ);

  IF $s \in$ *Post*(s')

    THEN ~~*Push*(ξ, s)~~; return "true"

Algorithm *CYCLE_CHECK*(s)

Push(ξ, s); insert s in V;

WHILE $\xi \neq \varnothing$ DO

  $s' :=$ *Top*(ξ);

  IF $s \in$ *Post*(s')

    THEN *Push*(ξ, s); return "true"

    ELSE IF *Post*(s') $\not\subseteq V$

Algorithm *CYCLE_CHECK*(s)

Push(ξ, s); insert s in V;

WHILE $\xi \neq \varnothing$ DO

  $s' :=$ *Top*(ξ);

  IF $s \in$ *Post*(s')

    THEN *Push*(ξ, s); return "true"

    ELSE IF *Post*(s') $\not\subseteq V$

      THEN choose $s'' \in$ *Post*(s') $\setminus V$;

        insert s'' in V; *Push*(ξ, s'');

Algorithm *CYCLE_CHECK*(s)

$Push(\xi, s)$; insert s in V;

WHILE $\xi \neq \varnothing$ DO

  $s' := Top(\xi)$;

  IF $s \in Post(s')$

    THEN $Push(\xi, s)$; return "true"

    ELSE IF $Post(s') \not\subseteq V$

      THEN choose $s'' \in Post(s') \setminus V$;

        insert s'' in V; $Push(\xi, s'')$;

      ELSE $Pop(\xi)$

    FI

Algorithm *CYCLE_CHECK*(s)

Push(ξ, s); insert s in V;
WHILE $\xi \neq \varnothing$ DO
    $s' :=$ *Top*(ξ);
    IF $s \in$ *Post*(s')
        THEN *Push*(ξ, s); return "true"
        ELSE IF *Post*(s') $\not\subseteq V$
                THEN choose $s'' \in$ *Post*(s') $\setminus V$;
                     insert s'' in V; *Push*(ξ, s'');
                ELSE *Pop*(ξ)
             FI
    FI
OD
return "false"

Algorithm *CYCLE_CHECK*(s)


```
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"
```

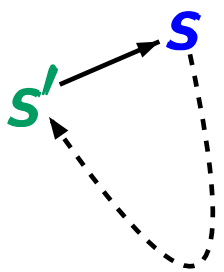
Algorithm *CYCLE_CHECK*(s)

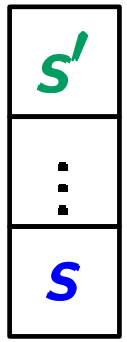
Push(ξ, s); insert s in V;

WHILE $\xi \neq \varnothing$ DO

$s' :=$ *Top*(ξ);

IF $s \in$ *Post*(s')





stack ξ

THEN *Push*(ξ, s); return "true"

ELSE IF *Post*(s') $\not\subseteq V$

THEN choose $s'' \in$ *Post*(s') $\setminus V$;

insert s'' in V; *Push*(ξ, s'');

ELSE *Pop*(ξ)

FI

FI

OD

return "false"

Algorithm *CYCLE_CHECK*(s)

Push(ξ, s); insert s in V;

WHILE $\xi \neq \varnothing$ DO

$s' :=$ *Top*(ξ);

IF $s \in$ *Post*(s')

THEN *Push*(ξ, s) return "true"

ELSE IF *Post*(s') $\not\subseteq V$

THEN choose $s'' \in$ *Post*(s') $\setminus V$;

insert s'' in V; *Push*(ξ, s'');

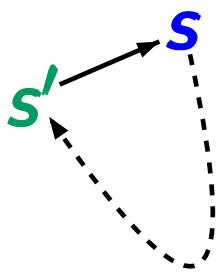
ELSE *Pop*(ξ)

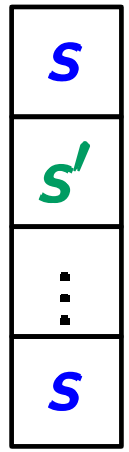
FI

FI

OD

return "false"





stack ξ

Nested DFS with counterexample generation

$U := \emptyset$; $\pi := \emptyset$; $V := \emptyset$; $\xi := \emptyset$;
WHILE $S_0 \not\subseteq U$ DO
    choose $s_0 \in S_0 \setminus U$; insert s_0 in U; $Push(\pi, s_0)$;
    WHILE $\pi \neq \emptyset$ DO
        $s := Top(\pi)$;
        IF $Post(s) \not\subseteq U$
         THEN choose $s' \in Post(s) \setminus U$;
              insert s' in U; $Push(\pi, s')$
         ELSE $Pop(\pi)$;
              IF $s \not\models a$ and $CYCLE\_CHECK(s)$
                 THEN return "**no**" + $reverse(\pi, \xi)$ FI
        FI
    OD
OD
return "**yes**"

Nested DFS with counterexample generation

$U := \emptyset$; $\pi := \emptyset$; $V := \emptyset$; $\xi := \emptyset$;
WHILE $S_0 \not\subseteq U$ DO
choose $s_0 \in S_0 \setminus U$; insert s_0 in U; $Push(\pi, s_0)$;
WHILE $\pi \neq \emptyset$ DO
$s := Top(\pi)$;
IF $Post(s) \not\subseteq U$
THEN choose $s' \in Post(s) \setminus U$;
insert s' in U; $Push(\pi, s')$
ELSE $Pop(\pi)$;
IF $s \not\models a$ and $CYCLE\_CHECK(s)$
THEN return "**no**" + $reverse(\pi, \xi)$ FI
FI
OD
OD
return "**yes**"

$DFS(s)$ starts when s inserted in U

$\longleftarrow$ $DFS(s)$ ends

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm *CYCLE_CHECK*(s)

- is called for $s \not\models a$ when *DFS*(s) is finished
- uses global data structures V and ξ

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm *CYCLE_CHECK*(s)

- is called for $s \not\models a$ when *DFS*(s) is finished
- uses global data structures V and ξ

V: organizes all states that have been visited in the current and all previous calls of *CYCLE_CHECK*($\cdot$)

ξ: stack for counterexample

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm *CYCLE_CHECK*(s)

- is called for $s \not\models a$ when *DFS*(s) is finished
- uses global data structures V and ξ

V: organizes all states that have been visited in the current and all previous calls of *CYCLE_CHECK*($\cdot$)

ξ: stack for counterexample

soundness:
1. termination
2. partial correctness

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm *CYCLE_CHECK*(s)

- is called for $s \not\models a$ when *DFS*(s) is finished
- uses global data structures V and ξ

V: organizes all states that have been visited in the current and all previous calls of *CYCLE_CHECK*($\cdot$)

ξ: stack for counterexample

soundness: 1. termination ✓

2. partial correctness

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm *CYCLE_CHECK*(s)

- is called for $s \not\models a$ when *DFS*(s) is finished
- uses global data structures V and ξ

V: organizes all states that have been visited in the current and all previous calls of *CYCLE_CHECK*($\cdot$)

ξ: stack for counterexample

$\mathcal{T} \models$ "eventually forever a"

iff the nested DFS returns "**yes**"

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm *CYCLE_CHECK*(s)

- is called for $s \not\models a$ when *DFS*(s) is finished
- uses global data structures V and ξ

V: organizes all states that have been visited in the current and all previous calls of *CYCLE_CHECK*($\cdot$)

ξ: stack for counterexample

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

iff the nested DFS returns "**no**"

Soundness of the nested DFS

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

iff the nested DFS returns "**no**"

Soundness of the nested DFS

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

iff the nested DFS returns "**no**"

Proof of "$\Longleftarrow$":

Soundness of the nested DFS

$\mathcal{T} \not\models$ "eventually forever a"
iff the nested DFS returns "**no**"

Proof of "$\Longleftarrow$":

If the nested DFS returns "**no**" then there is a reachable state s such that $s \not\models a$ and *CYCLE_CHECK*(s) finds a backward edge $t \longrightarrow s$.

Soundness of the nested DFS

$\mathcal{T} \not\models$ "eventually forever a"
iff the nested DFS returns "**no**"

Proof of "$\Longleftarrow$":

If the nested DFS returns "**no**" then there is a reachable state s such that $s \not\models a$ and *CYCLE_CHECK*(s) finds a backward edge $t \longrightarrow s$.

Hence: s belongs to a cycle

Soundness of the nested DFS

$\mathcal{T} \not\models$ "eventually forever a"
iff the nested DFS returns "**no**"

Proof of "$\Longleftarrow$":

If the nested DFS returns "**no**" then there is a reachable state s such that $s \not\models a$ and *CYCLE_CHECK*(s) finds a backward edge $t \longrightarrow s$.

Hence: s belongs to a cycle and there is an ultimatively periodic path $\pi = s_0 \ldots s_{n-1} (s\, t_1 \ldots t_k)^\omega$

Soundness of the nested DFS

$\mathcal{T} \not\models$ "eventually forever a"
iff the nested DFS returns "**no**"

Proof of "$\Longleftarrow$":

If the nested DFS returns "**no**" then there is a reachable state s such that $s \not\models a$ and *CYCLE_CHECK*(s) finds a backward edge $t \longrightarrow s$.

Hence: s belongs to a cycle and there is an ultimatively periodic path $\pi = s_0 \ldots s_{n-1} (s\, t_1 \ldots t_k)^\omega$ in $\mathcal{T}$ s.t. $\mathit{trace}(\pi) \notin$ "eventually forever a"

Soundness of the nested DFS

$\mathcal{T} \not\models$ "eventually forever a"
iff the nested DFS returns "**no**"

Proof of "$\Longleftarrow$":

If the nested DFS returns "**no**" then there is a reachable state s such that $s \not\models a$ and *CYCLE_CHECK*(s) finds a backward edge $t \longrightarrow s$.

Hence: s belongs to a cycle and there is an ultimatively periodic path $\pi = s_0 \ldots s_{n-1} (s\, t_1 \ldots . t_k)^\omega$ in $\mathcal{T}$ s.t. $\mathit{trace}(\pi) \notin$ "eventually forever a"

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

Soundness of the nested DFS

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

iff the nested DFS returns "**no**"

Proof of "$\Longrightarrow$":

Soundness of the nested DFS

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

iff the nested DFS returns "**no**"

Proof of "$\Longrightarrow$": show that:

When *CYCLE_CHECK*(s) is called then there is no cycle $t_0\, t_1 \ldots t_k$ in $\mathcal{T}$ s.t. $s = t_0 = t_k$ and $t_i \in V$ for some $i \in \{1, \ldots, k\}$

global visiting set of the inner DFS

Soundness of the nested DFS

$\mathcal{T} \not\models$ "eventually forever a"
iff the nested DFS returns "**no**"

Proof of "$\Longrightarrow$": show that:

When *CYCLE_CHECK*(s) is called then there is no cycle $t_0\, t_1 \ldots t_k$ in $\mathcal{T}$ s.t. $s = t_0 = t_k$ and

$t_i \in V$ for some $i \in \{1, \ldots, k\}$

global visiting set of the inner DFS

Hence: if s belongs to a cycle then *CYCLE_CHECK*(s) will find a backward edge $t \longrightarrow s$

Further improvements of the nested DFS

Further improvements of the nested DFS

$U := \emptyset$; $\pi := \emptyset$;
$V := \emptyset$; $\xi := \emptyset$;
WHILE $S_0 \not\subseteq U$ DO
    choose $s_0 \in S_0 \setminus U$;
        ⋮
    WHILE $\pi \neq \emptyset$ DO
        $s := Top(\pi)$;
        IF $Post(s) \not\subseteq U$
            THEN ...
            ELSE $Pop(\pi)$;
             IF $s \not\models a$ and $CYCLE\_CHECK(s)$ THEN ...
        FI
    OD
OD

Further improvements of the nested DFS

$U := \emptyset$; $\pi := \emptyset$;
$V := \emptyset$; $\xi := \emptyset$;
WHILE $S_0 \not\subseteq U$ DO
    choose $s_0 \in S_0 \setminus U$;
        ⋮
    WHILE $\pi \neq \emptyset$ DO
        $s := Top(\pi)$;
        IF $Post(s) \not\subseteq U$
            THEN ...
            ELSE $Pop(\pi)$;
                IF $s \not\models a$ and $CYCLE\_CHECK(s)$ THEN ...
        FI
    OD
OD

on the fly construction

Further improvements of the nested DFS

$U := \emptyset$; $\pi := \emptyset$;
$V := \emptyset$; $\xi := \emptyset$;
WHILE $S_0 \not\subseteq U$ DO
    choose $s_0 \in S_0 \setminus U$;
        ⋮
    WHILE $\pi \neq \emptyset$ DO
        $s := Top(\pi)$;
        IF $Post(s) \not\subseteq U$
            THEN ...
            ELSE $Pop(\pi)$;
             IF $s \not\models a$ and $CYCLE\_CHECK(s)$ THEN ...
        FI
    OD
OD

on the fly construction
hash techniques for U and V

Further improvements of the nested DFS

$U := \emptyset$; $\pi := \emptyset$;
$V := \emptyset$; $\xi := \emptyset$;
WHILE $S_0 \not\subseteq U$ DO
    choose $s_0 \in S_0 \setminus U$;
        ⋮
    WHILE $\pi \neq \emptyset$ DO
        $s := Top(\pi)$;
        IF $Post(s) \not\subseteq U$
            THEN ...
            ELSE $Pop(\pi)$;
             IF $s \not\models a$ and $CYCLE\_CHECK(s)$ THEN ...
    FI
OD
OD

on the fly construction
hash techniques for U and V

$s \in U \setminus V$: $\langle s, 1 \rangle$ $\quad$ $s \in V$: $\langle s, 0 \rangle$

Further improvements of the nested DFS

$U := \emptyset$; $\pi := \emptyset$;
$V := \emptyset$; $\xi := \emptyset$;
WHILE $S_0 \not\subseteq U$ DO
  choose $s_0 \in S_0 \setminus U$;
    ⋮
  WHILE $\pi \neq \emptyset$ DO
    $s := Top(\pi)$;
    IF $Post(s) \not\subseteq U$
      THEN ...
      ELSE $Pop(\pi)$;
        IF $s \not\models a$ and $CYCLE\_CHECK(s)$ THEN ...
    FI
  OD
OD

on the fly construction

hash techniques for U and V

$$s \in \underbrace{U \setminus V}_{\langle s,1\rangle} \qquad \underbrace{s \in V}_{\langle s,0\rangle}$$

early termination of $CYCLE\_CHECK$, e.g., if a state in π is visited