Overview

Introduction

Modelling parallel systems

Linear Time Properties

Regular Properties

- regular safety properties
- ω-regular properties
- model checking with Büchi automata $\longleftarrow$

Linear Temporal Logic

Computation-Tree Logic

Equivalences and Abstraction

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.,

$$\mathcal{L}_\omega(\mathcal{A}) \; = \; \left(2^{AP}\right)^\omega \setminus E$$

(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.,
$\mathcal{L}_\omega(\mathcal{A}) \; = \; \left(2^{AP}\right)^\omega \setminus E$

(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

$$\mathcal{T} \otimes \mathcal{A} \models \text{“never acceptance condition of } \mathcal{A}\text{”}$$

requires techniques for checking **persistence properties** in finite TS

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

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

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

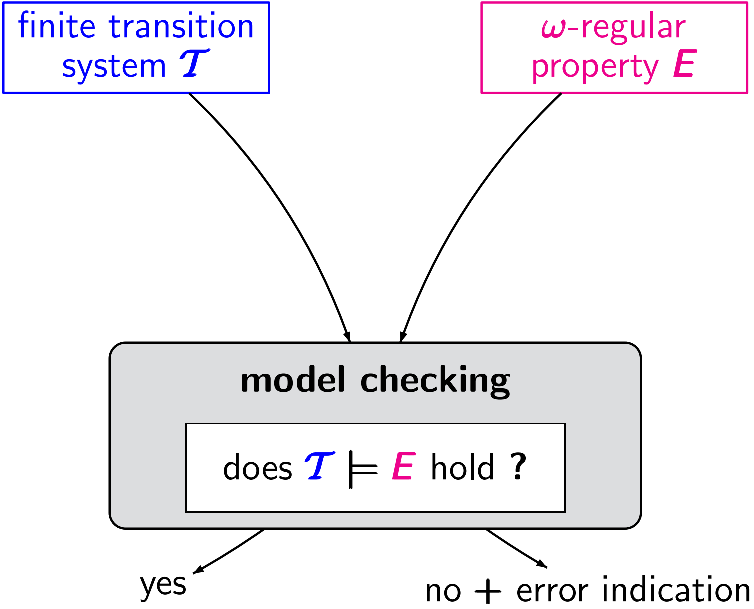
> E is called a persistence property if there exists a propositional formula Φ over AP such that
>
> $$E = \left\{ \begin{array}{l} \text{set of all infinite words } A_0\, A_1\, A_2 \ldots \in \left(2^{AP}\right)^{\omega} \\ \qquad\qquad \text{s.t.} \quad \overset{\infty}{\forall}\, i \geq 0.\ A_i \models \Phi \end{array} \right.$$

"from some moment on Φ"

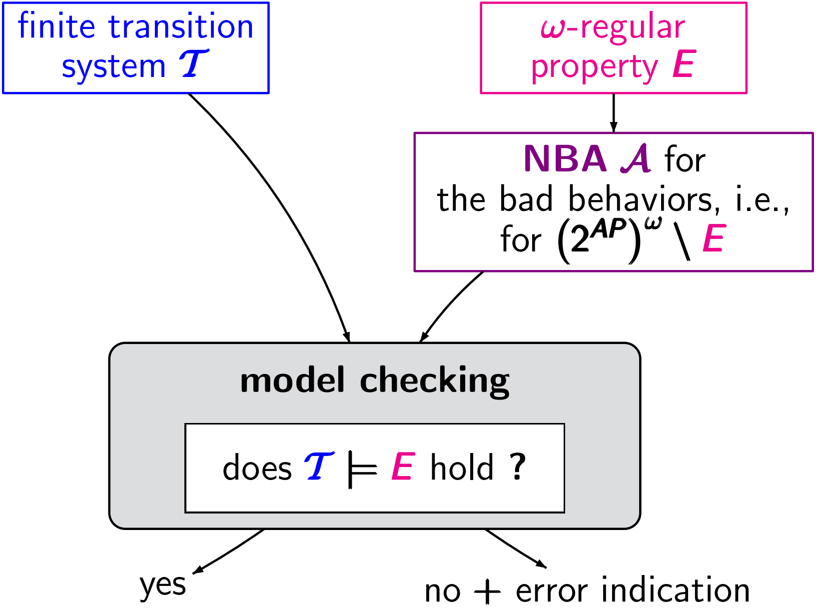
"eventually forever Φ"

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

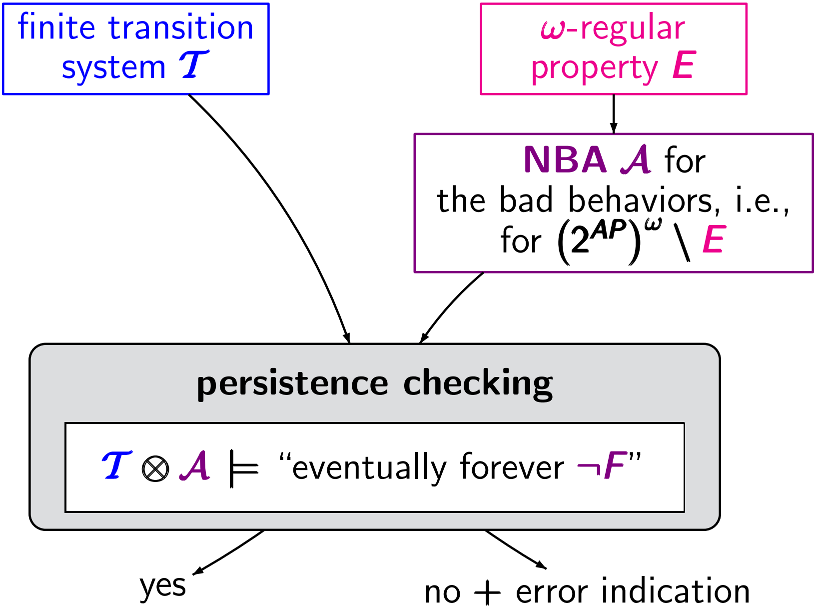
Checking ω-regular properties



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

$$\begin{array}{c} s_0 \\ \downarrow \\ s_1 \\ \downarrow \\ s_2 \\ \downarrow \\ \vdots \\ \downarrow \\ s_n \end{array}$$

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

$$\begin{array}{ccc}
s_0 & & L(s_0)=A_0 \\
\downarrow & & \\
s_1 & & L(s_1)=A_1 \\
\downarrow & & \\
s_2 & & L(s_2)=A_2 \\
\downarrow & & \vdots \\
\vdots & & \\
\downarrow & & \\
s_n & & L(s_n)=A_n
\end{array}$$

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

$$\begin{array}{ccc}
s_0 & & L(s_0) = A_0 \\
\downarrow & & \\
s_1 & & L(s_1) = A_1 \\
\downarrow & & \\
s_2 & & L(s_2) = A_2 \\
\downarrow & & \\
\vdots & & \vdots \\
\downarrow & & \\
s_n & & L(s_n) = A_n
\end{array}$$

path fragment $\widehat{\pi}$ trace

$$\begin{array}{c}
q_0 \in Q_0 \\
\downarrow A_0 \\
q_1 \\
\downarrow A_1 \\
q_2 \\
\downarrow A_2 \\
\vdots \\
\downarrow \\
q_n \\
\downarrow A_n \\
q_{n+1}
\end{array}$$

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

path fragment $\widehat{\pi}$	trace	path fragm. in product	run for $trace(\widehat{\pi})$
			$q_0 \in Q_0$
s_0	$L(s_0)=A_0$	$\langle s_0, q_1 \rangle$	$q_0 \xrightarrow{A_0} q_1$
s_1	$L(s_1)=A_1$	$\langle s_1, q_2 \rangle$	$q_1 \xrightarrow{A_1} q_2$
s_2	$L(s_2)=A_2$	$\langle s_2, q_3 \rangle$	$q_2 \xrightarrow{A_2} \vdots$
$\vdots$	$\vdots$	$\vdots$	q_n
s_n	$L(s_n)=A_n$	$\langle s_n, q_{n+1} \rangle$	$q_n \xrightarrow{A_n} q_{n+1}$

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

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

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 $\longleftarrow$ $\boxed{\text{same definition for } \textbf{NBA}}$

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

set of atomic propositions: $AP' = Q$

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

Product of a TS and NBA

$$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L) \quad \text{transition system}$$
$$\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F) \quad \textbf{NFA} \text{ or } \textbf{NBA}$$

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

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 ?

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

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

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ 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., $\mathcal{L}_\omega(\mathcal{A}) = \left(2^{AP}\right)^\omega \setminus E$

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ 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) $Traces(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

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

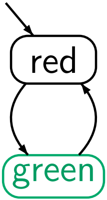
(1) $\mathcal{T} \models E$

(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

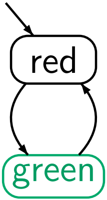
TS $\mathcal{T}$



LT property: "infinitely often green"

Example: ω-regular model checking

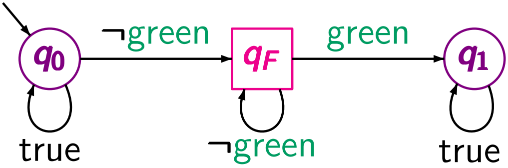
TS $\mathcal{T}$



LT property: "infinitely often green"

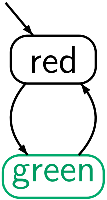
NBA $\mathcal{A}$ for the complement

"from some moment on $\neg$green"



Example: ω-regular model checking

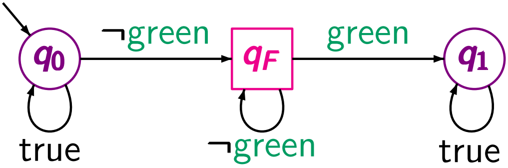
TS $\mathcal{T}$

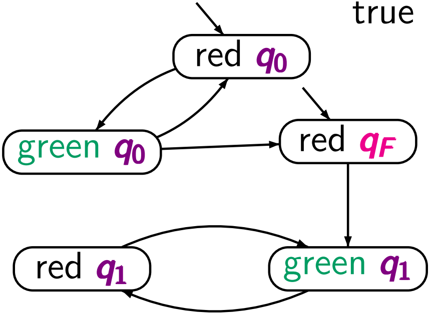


LT property: "infinitely often green"

NBA $\mathcal{A}$ for the complement

"from some moment on $\neg$green"

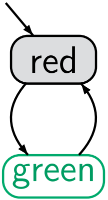




reachable fragment of the product TS $\mathcal{T} \otimes \mathcal{A}$

Example: ω-regular model checking

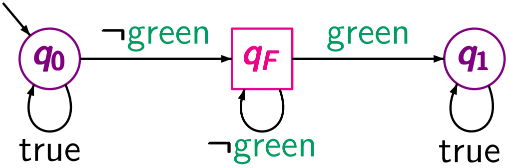
TS $\mathcal{T}$

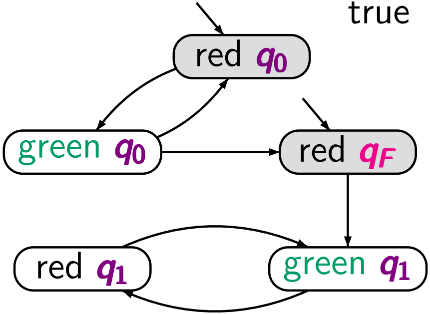


LT property: "infinitely often green"

NBA $\mathcal{A}$ for the complement

"from some moment on $\neg$green"



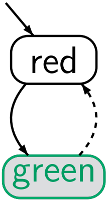


initial states:

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

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

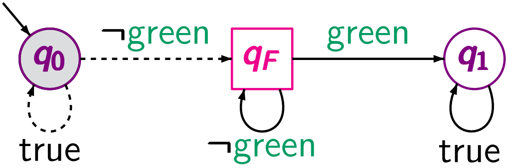
TS $\mathcal{T}$

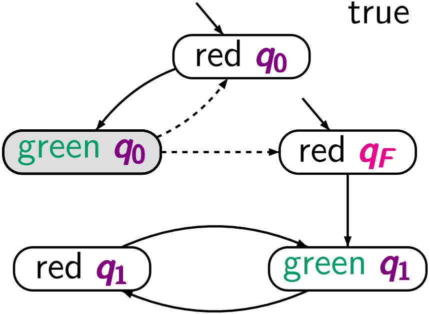


LT property: "infinitely often green"

NBA $\mathcal{A}$ for the complement

"from some moment on $\neg$green"





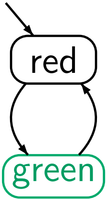
transition

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

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

Example: ω-regular model checking

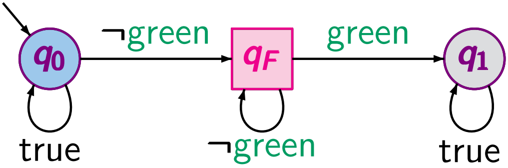
TS $\mathcal{T}$

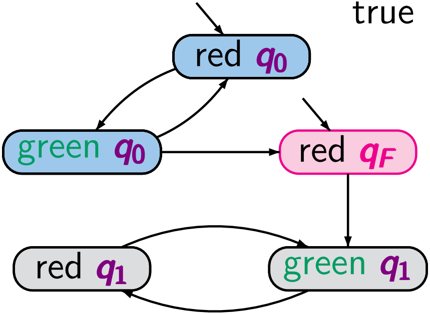


LT property: "infinitely often green"

NBA $\mathcal{A}$ for the complement

"from some moment on $\neg$green"





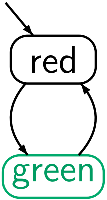
atomic propositions

$AP' = \{q_0, q_F, q_1\}$

obvious labeling function

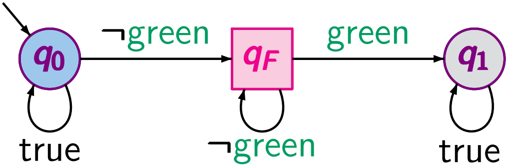
Example: ω-regular model checking

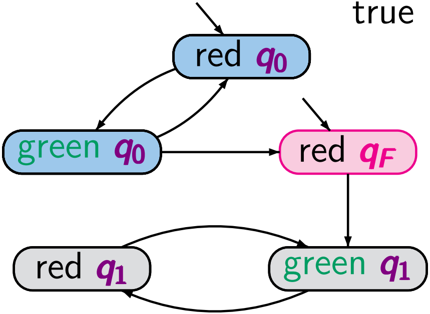
TS $\mathcal{T}$



LT property: "infinitely often green"

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





atomic propositions

$AP' = \{q_0, q_F, q_1\}$

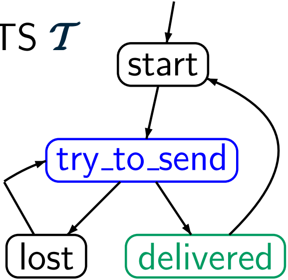
obvious labeling function

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

"eventually forever $\neg F$"

Example: ω-regular model checking

TS $\mathcal{T}$

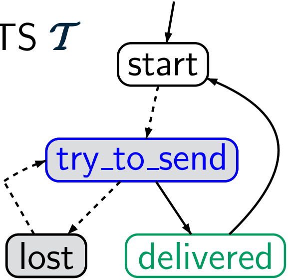


ω-regular LT property E:

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

Example: ω-regular model checking

TS $\mathcal{T}$



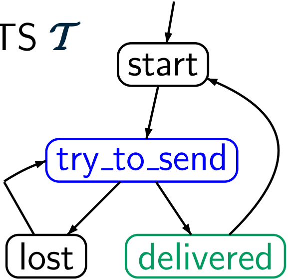
ω-regular LT property E:

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

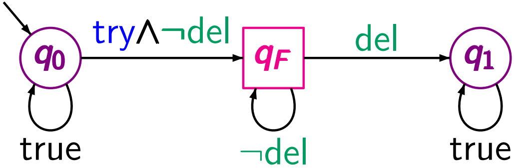
$\mathcal{T} \not\models E$

Example: ω-regular model checking

TS $\mathcal{T}$



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



ω-regular LT property E:

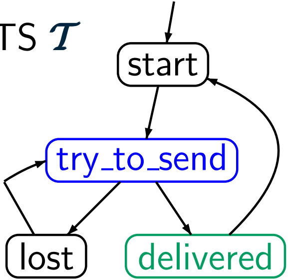
"each (repeatedly) sent message will eventually be delivered"

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

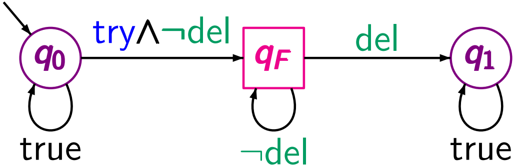
"never delivered after some trial"

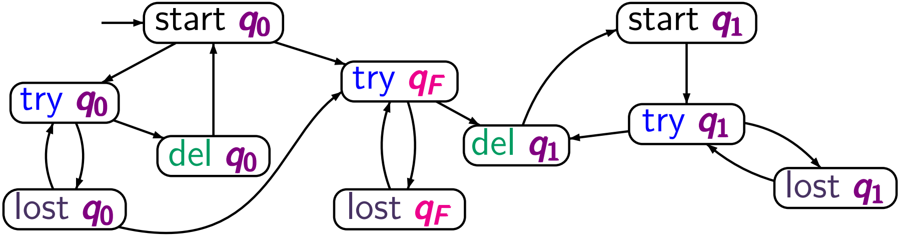
Example: ω-regular model checking

TS $\mathcal{T}$



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

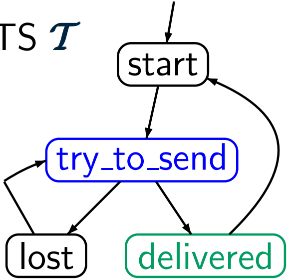




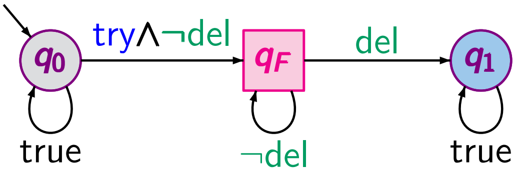
reachable fragment of the product-TS

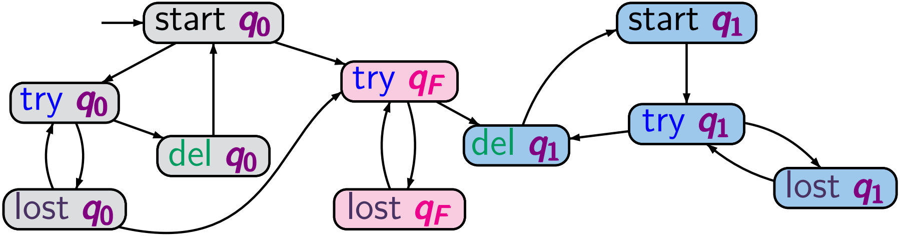
Example: ω-regular model checking

TS $\mathcal{T}$



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

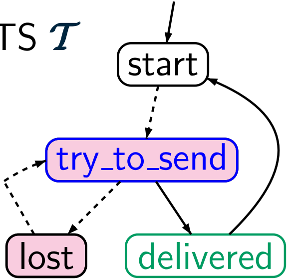




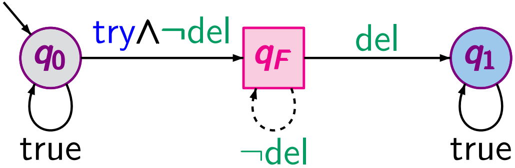
set of atomic propositions $AP' = \{q_0, q_1, q_F\}$

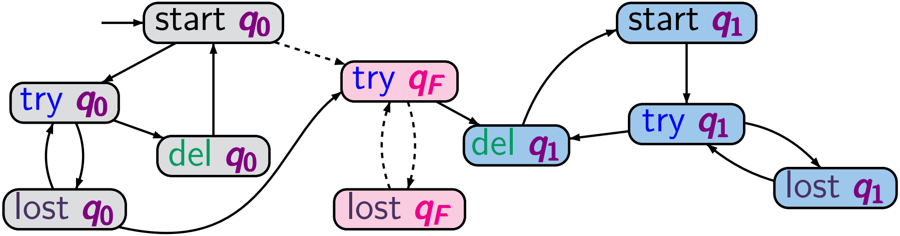
Example: ω-regular model checking

TS $\mathcal{T}$



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





$\mathcal{T} \otimes \mathcal{A} \not\models$ "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 $\quad Traces_{fin}(\mathcal{T}) \cap BadPref = \varnothing$

Checking safety and ω-regular properties

for regular safety property E

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

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

for ω-regular property E

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

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

$\mathcal{A}$ 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 $\quad Traces_{fin}(\mathcal{T}) \cap \mathcal{L}(\mathcal{A}) = \varnothing$

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

for ω-regular property E

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

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

$\mathcal{A}$ 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 $\quad Traces_{fin}(\mathcal{T}) \cap \mathcal{L}(\mathcal{A}) = \varnothing$

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

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

for ω-regular property E

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

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

iff $\quad \mathcal{T} \otimes \mathcal{A} \models$ "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 $s \ldots s$

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 $C \cap \{s \in S : s \not\models a\} \neq \varnothing$

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 $s \dots s$

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

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 $s \dots s$

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

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 ?

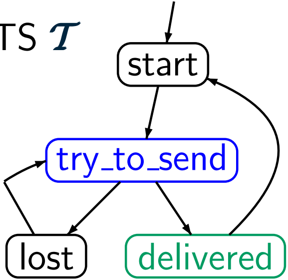
$\mathcal{T} \not\models$ "eventually forever a"

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 $C \cap \{s \in S : s \not\models a\} \neq \varnothing$

method: calculate and analyze the **SCCs**

TS $\mathcal{T}$

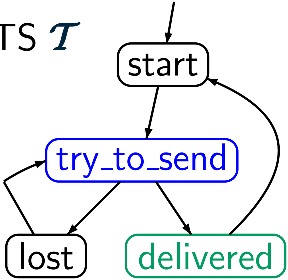


ω-regular LT property E:

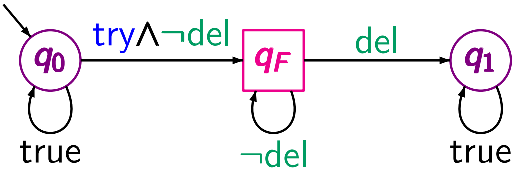
"each (repeatedly) sent message will eventually be delivered"

Example: ω-regular model checking

TS $\mathcal{T}$



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

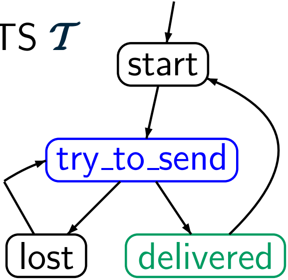


ω-regular LT property E:

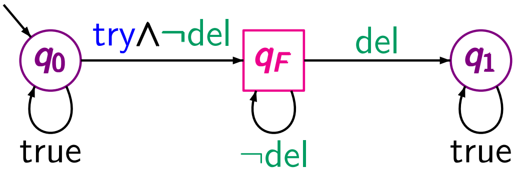
"each (repeatedly) sent message will eventually be delivered"

Example: ω-regular model checking

TS $\mathcal{T}$



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

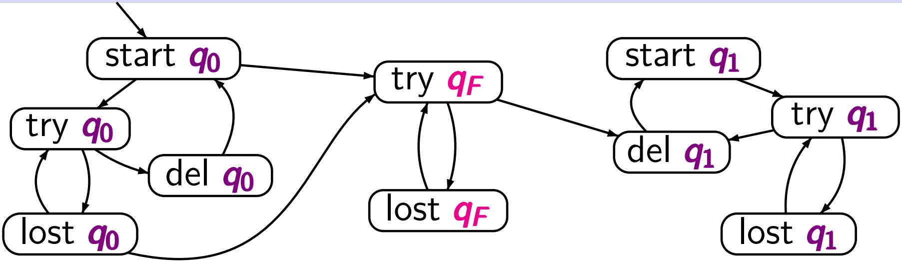


ω-regular LT property E:

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

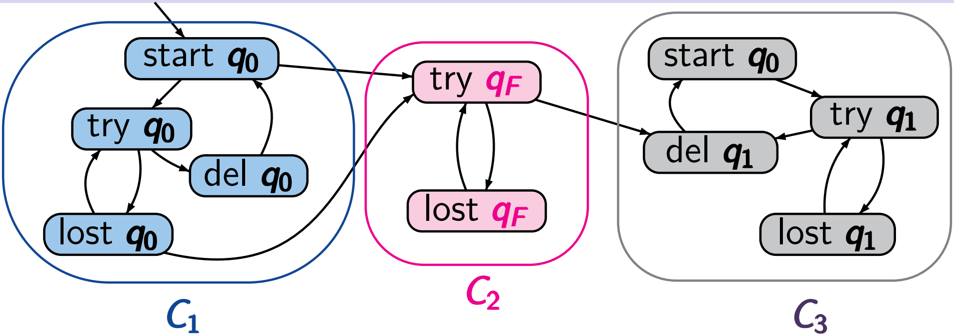
... analysis of the **SCCs** in product $\mathcal{T} \otimes \mathcal{A}$...

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



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

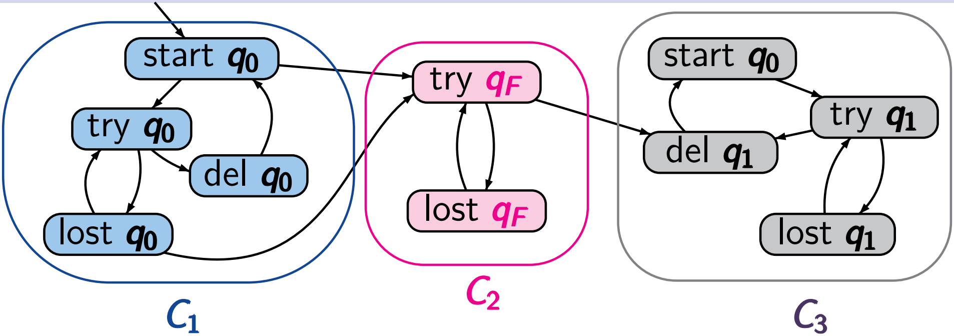
Example: persistence checking $\mathcal{T} \otimes \mathcal{A}$



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

3 reachable SCCs: C_1, C_2, C_3

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

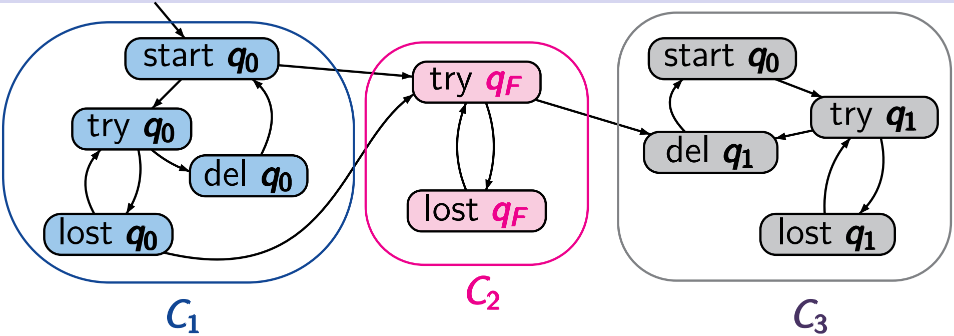


persistence property: "eventually forever $\neg q_F$"

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 $s \not\models \neg q_F$

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

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 $s \dots s$

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

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

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

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

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

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

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 ⊭ 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`

`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 $s \not\models a$;

mark s;

IF *CYCLE_CHECK*(s) THEN

return "no"

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

FI

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

return "yes"

Persistence checking $\longleftarrow$ 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 $s' \longrightarrow s$

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

return "yes"

Time complexity of nested DFS

Time complexity of nested DFS

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 $s' \to s$

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

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

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 $s' \to s$

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

return "yes"

worst case: $\Theta(|S| \cdot \underbrace{(|S| + \#\mathbf{edges})}_{\text{cost of } CYCLE\_CHECK(s) \text{ caused by each state } s \not\models a})$ naïve approach

$\boxed{\text{1. DFS: visits all reachable states}}$

REPEAT

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

mark s;

IF *CYCLE_CHECK*(s) THEN

return "no"

FI

$\boxed{\text{2. DFS: searches for a backward edge } s' \to s}$

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

return "yes"

worst case: $\Theta(|S| \cdot \underbrace{(|S| + \#\textbf{edges})}_{\text{cost of } CYCLE\_CHECK(s) \text{ caused by each state } s \not\models a})$ naïve approach

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

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 $s' \to s$

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

return "yes"

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

- 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 $s' \to s$

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

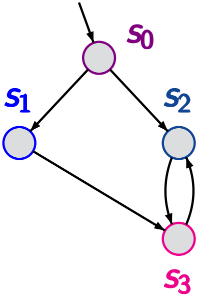
- serves to check whether $\mathcal{T} \models$ "eventually forever a"
- relies on two DFS running in an interleaved way

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 $s' \rightarrow s$
- 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**

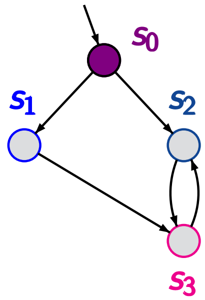


$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



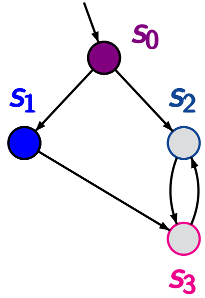
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

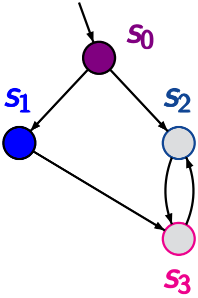
$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

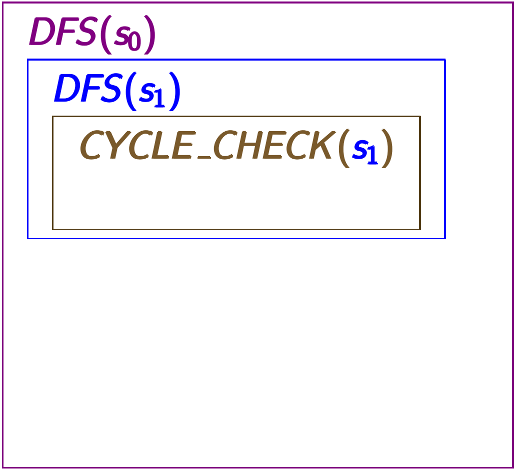
$\mathcal{T} \not\models$ "eventually forever a"

Example: nested DFS



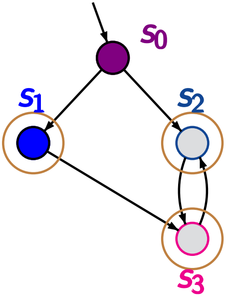
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$



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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

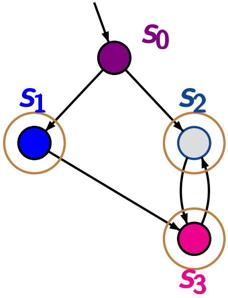
$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

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

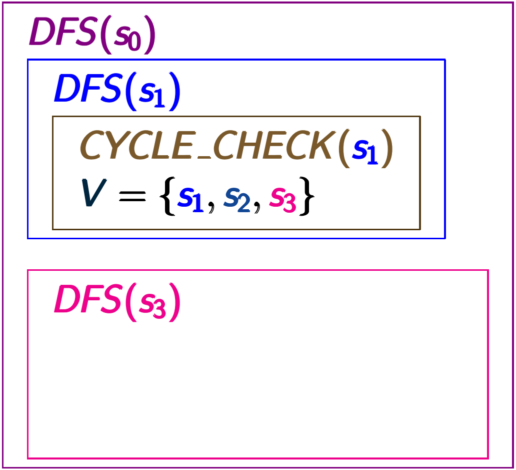
$\mathcal{T} \not\models$ "eventually forever a"

Example: nested DFS



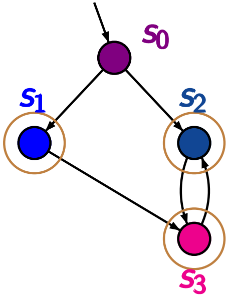
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$



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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

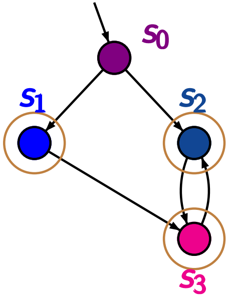
$V = \{s_1, s_2, s_3\}$

$DFS(s_3)$

$DFS(s_2)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

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

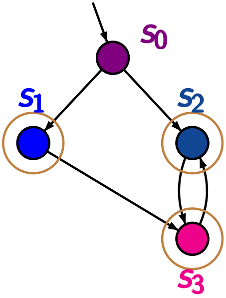
$DFS(s_3)$

$DFS(s_2)$

$CYCLE\_CHECK(s_2)$

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

Example: nested DFS ⟵ fails



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

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: $\mathit{CYCLE\_CHECK}(s)$ 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 $s' \to s$
- 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

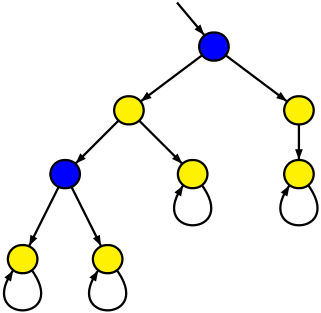
- serves to check whether $\mathcal{T} \models$ "eventually forever a"
- relies on two DFSs running in an interleaved way

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 $s' \to s$
- 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**

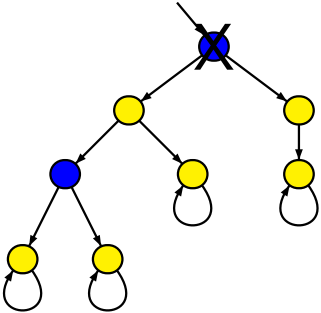
Nested DFS



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

1. DFS: visits all reachable states

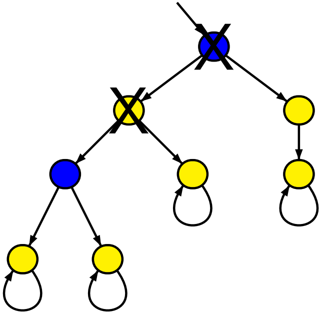
2. DFS: *CYCLE_CHECK*(s) for $s \models$ ***blue***
checks whether s belongs to a cycle
by searching a backward-edge $s' \to s$



$\mathcal{T} \models$ "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 $s' \longrightarrow s$

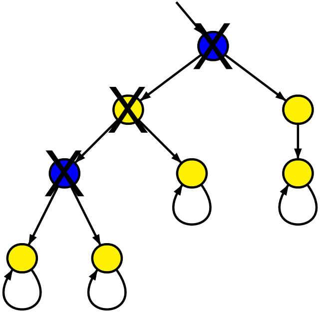
Nested DFS



$\mathcal{T} \models$ "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 $s' \rightarrow s$

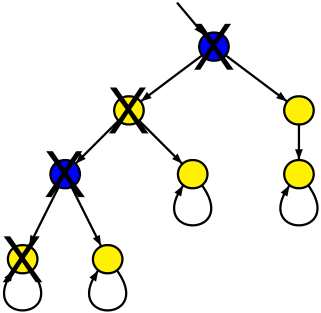
Nested DFS



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

1. DFS: visits all reachable states

2. DFS: $\mathit{CYCLE\_CHECK}(s)$ for $s \models$ ***blue***
checks whether s belongs to a cycle
by searching a backward-edge $s' \to s$

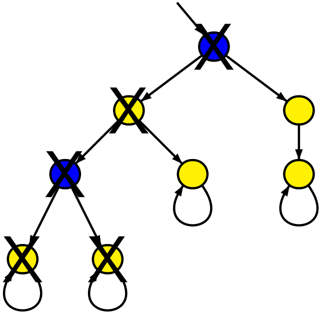


$\mathcal{T} \models$ "eventually forever $\neg$***blue***"

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 $s' \longrightarrow s$

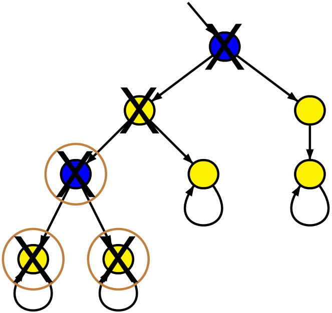
Nested DFS



$\mathcal{T} \models$ "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 $s' \to s$

Nested DFS

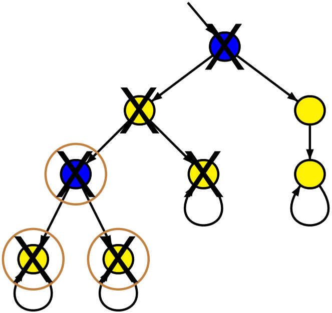


$\mathcal{T} \models$ "eventually forever $\neg$***blue***"

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

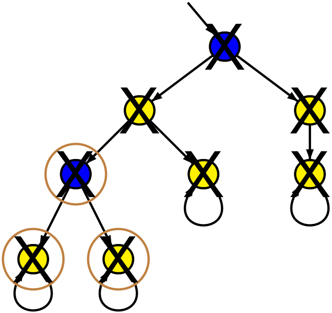


$\mathcal{T} \models$ "eventually forever $\neg$***blue***"

1. DFS: visits all reachable states

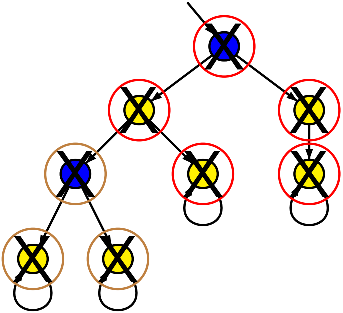
2. DFS: *CYCLE_CHECK*(s) for $s \models$ ***blue***
checks whether s belongs to a cycle
by searching a backward-edge $s' \rightarrow s$

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 $s' \rightarrow s$



$\mathcal{T} \models$ "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 (pseudo code)

$U := \varnothing$;

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

Nested DFS (pseudo code)

$U := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set of 1. DFS}}$

```
FOR ALL s_0 ∈ S_0 DO DFS(s_0) OD ;
```

Nested DFS (pseudo code)

$U := \varnothing$; $\longleftarrow$ 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)

$U := \varnothing$; $\longleftarrow$ 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 ;

Nested DFS (pseudo code)

$U := \varnothing$; $\longleftarrow$ visiting set of 1. DFS

```
FOR ALL s_0 ∈ S_0 DO DFS(s_0) 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)

$U := \varnothing$; $\longleftarrow$ visiting set of 1. DFS

$V := \varnothing$ $\longleftarrow$ global visiting set of 2. DFS

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

```
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
```

pseudo code for $DFS(s)$

Nested DFS (pseudo code)

$U := \varnothing$; $\longleftarrow$ visiting set of 1. DFS

$V := \varnothing$ $\longleftarrow$ 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)

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

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

```
FOR ALL s_0 ∈ S_0 DO DFS(s_0) 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)

$U := \varnothing$; $\longleftarrow$ visiting set of 1. DFS

$V := \varnothing$ $\longleftarrow$ global visiting set of 2. DFS

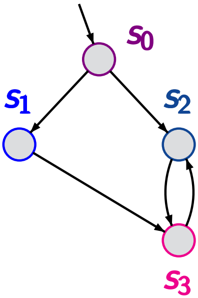
```
FOR ALL s_0 ∈ S_0 DO DFS(s_0) OD ;
return "yes"
```

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

```
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
```

pseudo code for $DFS(s)$

Example: nested DFS

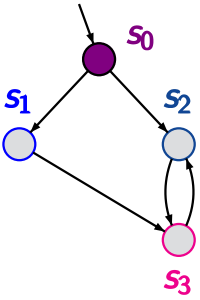


$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



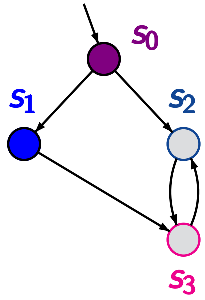
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

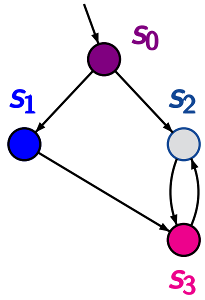
$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

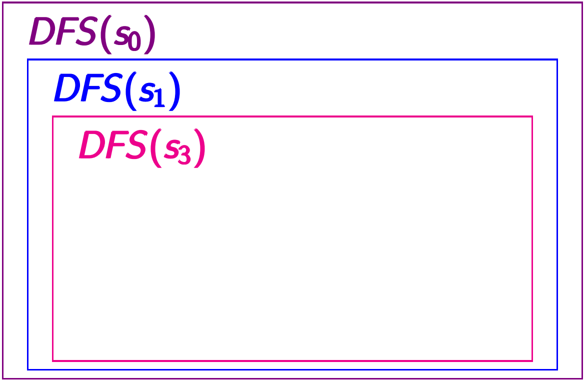
$\mathcal{T} \not\models$ "eventually forever a"

Example: nested DFS

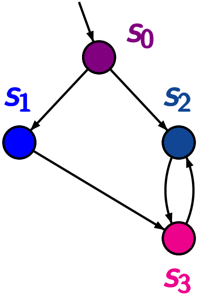


$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

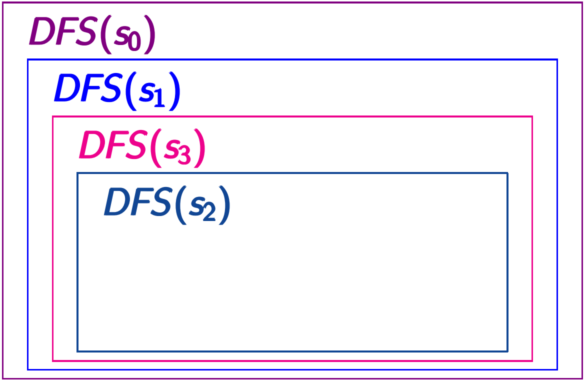


$\mathcal{T} \not\models$ "eventually forever a"



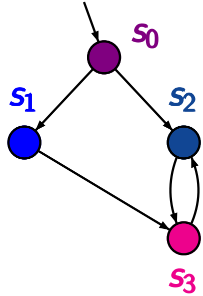
$s_1, s_2 \not\models a$

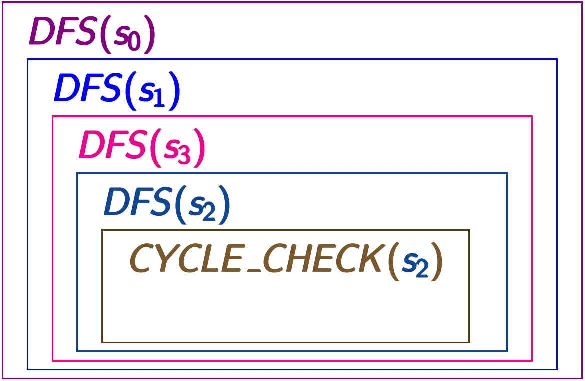
$s_0, s_3 \models a$



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

Example: nested DFS



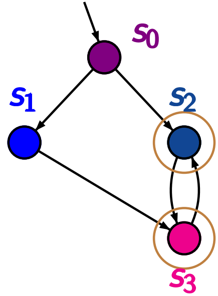


$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$DFS(s_3)$

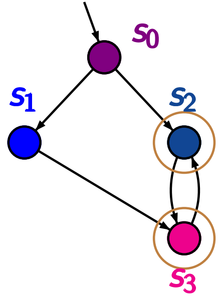
$DFS(s_2)$

$CYCLE\_CHECK(s_2)$

$\rightsquigarrow$ cycle $s_2\ s_3\ s_2$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

$DFS(s_0)$

$DFS(s_1)$

$DFS(s_3)$

$DFS(s_2)$

$CYCLE\_CHECK(s_2)$

$\rightsquigarrow$ cycle $s_2\ s_3\ s_2$

$\uparrow$

returns correct answer "**no**"

Nested DFS with counterexample generation

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

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

$\uparrow$

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 $s_n \not\models a$

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

$\uparrow$

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 ...

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

$U := \varnothing$; $\pi := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set and stack for 1. DFS}}$

$U := \varnothing$; $\pi := \varnothing$; $\leftarrow$ $\boxed{\text{visiting set and stack for 1. DFS}}$

$V := \varnothing$; $\xi := \varnothing$; $\leftarrow$ $\boxed{\text{visiting set and stack for 2. DFS}}$

$U := \varnothing$; $\pi := \varnothing$; $\leftarrow$ $\boxed{\text{visiting set and stack for 1. DFS}}$

$V := \varnothing$; $\xi := \varnothing$; $\leftarrow$ $\boxed{\text{visiting set and stack for 2. DFS}}$

WHILE $S_0 \not\subseteq U$ DO

OD

$U := \varnothing$; $\pi := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set and stack for 1. DFS}}$

$V := \varnothing$; $\xi := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set and stack for 2. DFS}}$

```
WHILE S_0 ⊈ U DO
    choose s_0 ∈ S_0 \ U; insert s_0 in U;
OD
```

$U := \varnothing;\ \pi := \varnothing;\quad \longleftarrow$ $\boxed{\text{visiting set and stack for 1. DFS}}$

$V := \varnothing;\ \xi := \varnothing;\quad \longleftarrow$ $\boxed{\text{visiting set and stack for 2. DFS}}$

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

$U := \varnothing$; $\pi := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set and stack for 1. DFS}}$

$V := \varnothing$; $\xi := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set and stack for 2. DFS}}$

```
WHILE S_0 ⊈ U DO
    choose s_0 ∈ S_0 \ U; insert s_0 in U; Push(π, s_0);
    WHILE π ≠ ∅ DO
        s := Top(π);



    OD
OD
```

$U := \varnothing;\ \pi := \varnothing;$ $\longleftarrow$ $\boxed{\text{visiting set and stack for 1. DFS}}$

$V := \varnothing;\ \xi := \varnothing;$ $\longleftarrow$ $\boxed{\text{visiting set and stack for 2. DFS}}$

```
WHILE S_0 ⊄ U DO
    choose s_0 ∈ S_0 \ U; insert s_0 in U; Push(π, s_0);
    WHILE π ≠ ∅ DO
        s := Top(π);
        IF Post(s) ⊄ U









        FI
    OD
OD
```

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

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

```
WHILE S_0 ⊈ U DO
    choose s_0 ∈ S_0 \ U; insert s_0 in U; Push(π, s_0);
    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
```

$U := \varnothing$; $\pi := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set and stack for 1. DFS}}$

$V := \varnothing$; $\xi := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set and stack for 2. DFS}}$

```
WHILE S_0 ⊄ U DO
    choose s_0 ∈ S_0 \ U; insert s_0 in U; Push(π, s_0);
    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
```

$U := \varnothing$; $\pi := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set and stack for 1. DFS}}$

$V := \varnothing$; $\xi := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set and stack for 2. DFS}}$

```
WHILE S_0 ⊈ U DO
    choose s_0 ∈ S_0 \ U; insert s_0 in U; Push(π, s_0);
    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 := ∅; π := ∅;   ← visiting set and stack for 1. DFS
V := ∅; ξ := ∅;   ← 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
```

$U := \varnothing$; $\pi := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set and stack for 1. DFS}}$

$V := \varnothing$; $\xi := \varnothing$; $\longleftarrow$ $\boxed{\text{visiting set and stack for 2. DFS}}$

```
WHILE S_0 ⊈ U DO
    choose s_0 ∈ S_0 \ U; insert s_0 in U; Push(π, s_0);
    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 $CYCLE\_CHECK(s)$

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

Algorithm $CYCLE\_CHECK(s)$

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

WHILE $\xi \neq \varnothing$ DO

Algorithm $CYCLE\_CHECK(s)$

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

WHILE $\xi \neq \varnothing$ DO

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

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

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 $\cancel{Push(\xi, s)}$; return "true"

Algorithm $CYCLE\_CHECK(s)$

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

WHILE $\xi \neq \varnothing$ DO

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

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

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

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

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; $Push(\xi, s'')$;

Algorithm $CYCLE\_CHECK(s)$

$Push(\xi, s)$; 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
```

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 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 $CYCLE\_CHECK(s)$

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

WHILE $\xi \neq \varnothing$ DO

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

  IF $\boxed{s \in Post(s')}$

    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; $Push(\xi, s'')$;

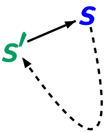
      ELSE $Pop(\xi)$

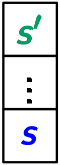
    FI

  FI

OD

return "false"





stack ξ

Algorithm $CYCLE\_CHECK(s)$

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

WHILE $\xi \neq \varnothing$ DO

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

$\quad$ IF $\boxed{s \in Post(s')}$

$\quad\quad$ THEN $\boxed{Push(\xi, s)}$ return "true"

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

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

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

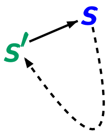
$\quad\quad\quad$ ELSE $Pop(\xi)$

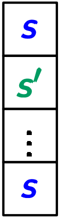
$\quad\quad$ FI

$\quad$ FI

OD

return "false"





stack ξ

```
U := ∅; π := ∅; V := ∅; ξ := ∅;
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"
```

```
U := ∅; π := ∅; V := ∅; ξ := ∅;
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);                     ⟵ DFS(s) ends
                  IF s ⊭ a and CYCLE_CHECK(s)
                    THEN return "no" + reverse(π, ξ) FI
            FI
      OD
OD
return "yes"
```

DFS(s) starts when s inserted in U

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 $\mathit{CYCLE\_CHECK}(s)$

- is called for $s \not\models a$ when $\mathit{DFS}(s)$ 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 $\mathit{CYCLE\_CHECK}(\cdot)$
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 $\surd$

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**"

$\mathcal{T} \not\models$ "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 $t \rightarrow s$.

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 $t \rightarrow s$.

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 $\pi = s_0 \dots s_{n-1}(\, s\, t_1 \dots .\, t_k\,)^\omega$

$\mathcal{T} \not\models$ "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 ultimately 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"

$\mathcal{T} \not\models$ "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 $t \rightarrow s$.

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

global visiting set of the inner 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 $t \to s$

Further improvements of the nested DFS

Further improvements of the nested DFS

```
U := ∅; π := ∅;
V := ∅; ξ := ∅;
WHILE S_0 ⊄ U DO
    choose s_0 ∈ S_0 \ U;
        ⋮
    WHILE π ≠ ∅ DO
        s := Top(π);
        IF Post(s) ⊄ U
            THEN ...
            ELSE Pop(π);
             IF s ⊭ a and CYCLE_CHECK(s) THEN ...
      FI
   OD
OD
```

Further improvements of the nested DFS

$U := \varnothing$; $\pi := \varnothing$;
$V := \varnothing$; $\xi := \varnothing$;
WHILE $S_0 \not\subseteq U$ DO
  choose $s_0 \in S_0 \setminus U$;
  ⋮
  WHILE $\pi \neq \varnothing$ 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

$U := \varnothing$; $\pi := \varnothing$;
$V := \varnothing$; $\xi := \varnothing$;
WHILE $S_0 \not\subseteq U$ DO
    choose $s_0 \in S_0 \setminus U$;
        ⋮
    WHILE $\pi \neq \varnothing$ 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 := ∅; π := ∅;
V := ∅; ξ := ∅;
WHILE S0 ⊈ U DO
    choose s0 ∈ S0 \ U;
        ⋮
    WHILE π ≠ ∅ DO
        s := Top(π);
        IF Post(s) ⊈ U
            THEN ...
            ELSE Pop(π);
             IF s ⊭ a and CYCLE_CHECK(s) THEN ...
        FI
    OD
OD
```

on the fly construction

hash techniques for U and V

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

Further improvements of the nested DFS

$U := \varnothing$; $\pi := \varnothing$;
$V := \varnothing$; $\xi := \varnothing$;
WHILE $S_0 \not\subseteq U$ DO
    choose $s_0 \in S_0 \setminus U$;
        ⋮
    WHILE $\pi \neq \varnothing$ 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

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

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