

Introduction

Modelling parallel systems

Linear Time Properties

## Regular Properties

regular safety properties

$\omega$ -regular properties

model checking with Büchi automata ←

Linear Temporal Logic

Computation-Tree Logic

Equivalences and Abstraction

# Verifying $\omega$ -regular properties

*given:* finite transition system  $\mathcal{T}$

$\omega$ -regular property  $E$

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

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

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

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

*given:* finite transition system  $\mathcal{T}$   
 $\omega$ -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  $\text{Traces}(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \emptyset$

# Verifying $\omega$ -regular properties

*given:* finite transition system  $\mathcal{T}$   
 $\omega$ -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  $\text{Traces}(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \emptyset$

(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 $\omega$ -regular properties

*given:* finite transition system  $\mathcal{T}$   
 $\omega$ -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  $\text{Traces}(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \emptyset$

(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



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  $\Phi$  over  $AP$  such that

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

$\forall i \geq 0. \dots = \exists j \geq 0 \forall i \geq j. \dots$  “for all but finitely many”



# Persistence property

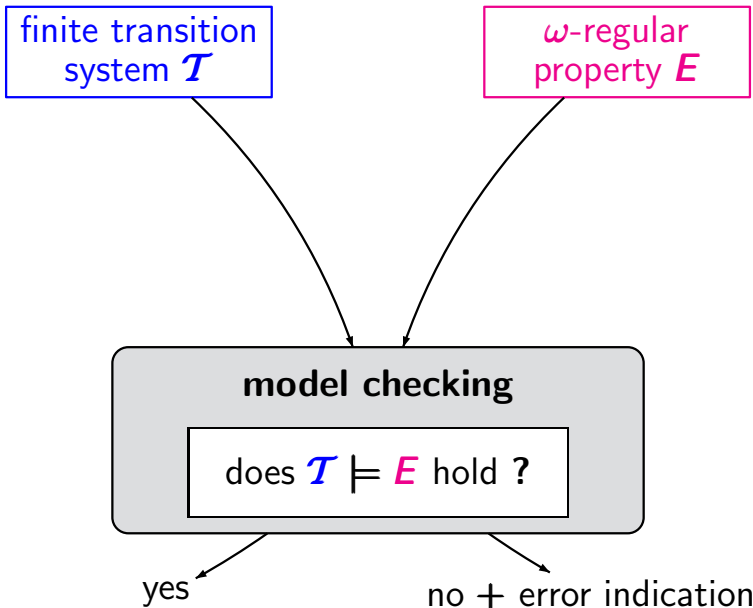
Let  $E$  be an LT-property, i.e.,  $E \subseteq (2^{AP})^\omega$

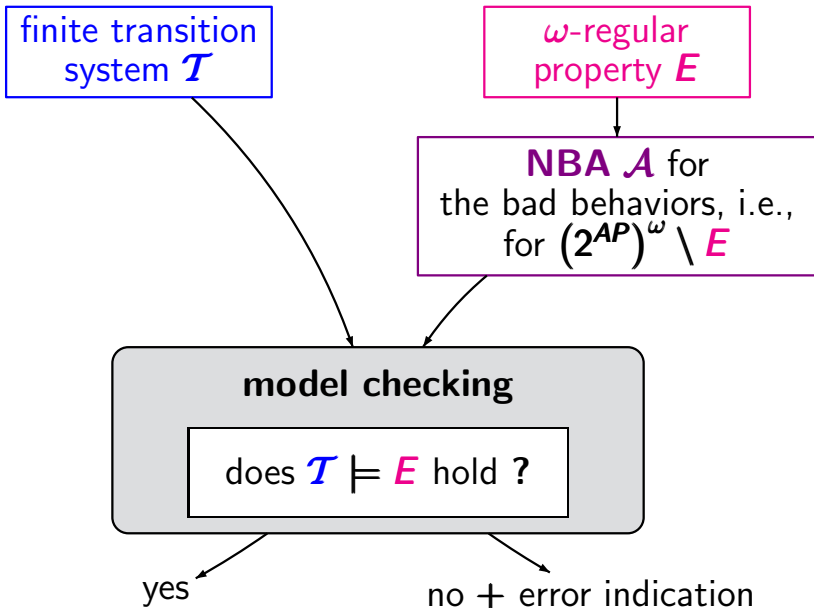
$E$  is called a **persistence property** if there exists a propositional formula  $\Phi$  over  $AP$  such that

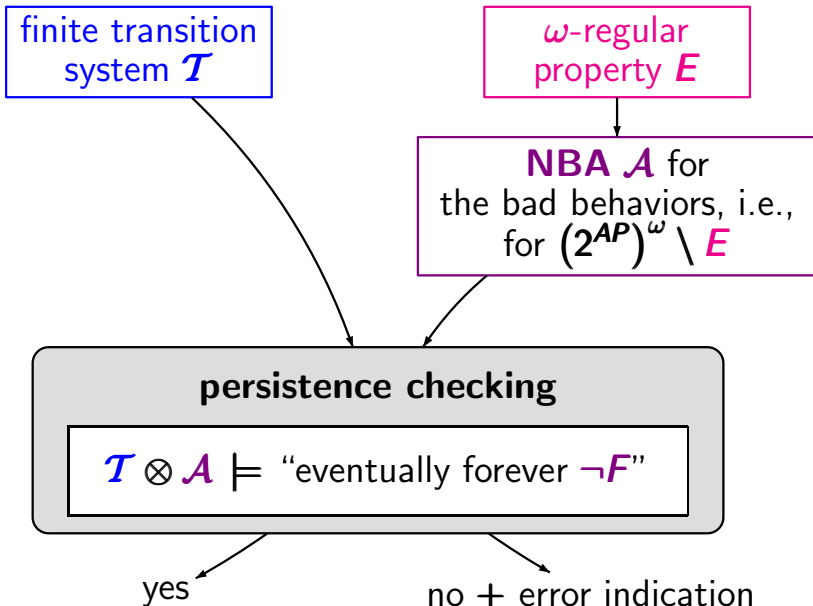
$$E = \left\{ \begin{array}{l} \text{set of all infinite words } A_0 A_1 A_2 \dots \in (2^{AP})^\omega \\ \text{s.t. } \forall i \geq 0. A_i \models \Phi \end{array} \right.$$

↑  
“from some moment on  $\Phi$ ”  
“eventually forever  $\Phi$ ”

$\forall i \geq 0. \dots = \exists j \geq 0 \forall i \geq j. \dots$  “for all but finitely many”







# Recall: product of a TS and an NFA

finite transition system

$$\mathcal{T} = (S, Act, \rightarrow, s_0, AP, L)$$

NFA for bad prefixes

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

 $s_0$  $s_1$  $s_2$  $\vdots$  $s_n$ 

path  
fragment  $\hat{\pi}$

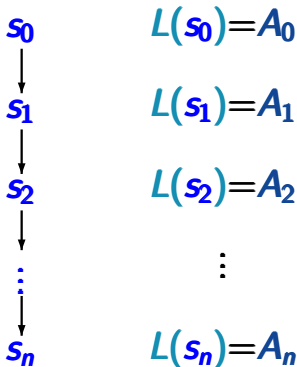
# Recall: product of a TS and an NFA

finite transition system

$$\mathcal{T} = (S, Act, \rightarrow, s_0, AP, L)$$

NFA for bad prefixes

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



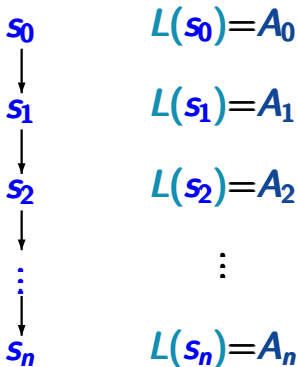
path  
fragment  $\hat{\pi}$

trace

# Recall: product of a TS and an NFA

finite transition system

$$\mathcal{T} = (\mathcal{S}, \text{Act}, \rightarrow, s_0, \text{AP}, L)$$

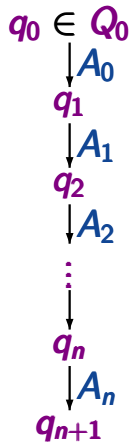


path  
fragment  $\hat{\pi}$

trace

NFA for bad prefixes

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



run for  $\text{trace}(\hat{\pi})$

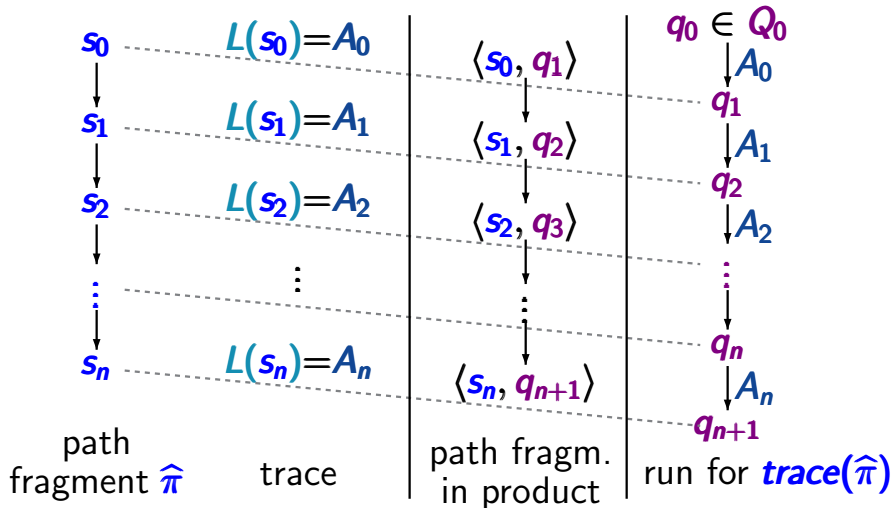
# Recall: product of a TS and an NFA

finite transition system

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

NFA for bad prefixes

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





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

$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$  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:  $L'(\langle s, q \rangle) = \{q\}$

*given:*        finite TS  $\mathcal{T}$   
                   $\omega$ -regular LT property  $E$

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

*given:*        finite TS  $\mathcal{T}$   
                   $\omega$ -regular LT property  $E$

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

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



*given:*        finite TS  $\mathcal{T}$   
                   $\omega$ -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**

- $\mathcal{T} = (\mathcal{S}, Act, \rightarrow, \mathcal{S}_0, AP, L)$  finite transition system  
without terminal states
- $\mathcal{A} = (\mathcal{Q}, 2^{AP}, \delta, \mathcal{Q}_0, F)$  non-blocking NBA  
representing the bad behaviors of an  $\omega$ -regular  
LT-property  $E$

$\mathcal{T} = (\mathcal{S}, Act, \rightarrow, \mathcal{S}_0, AP, L)$  finite transition system  
without terminal states

$\mathcal{A} = (\mathcal{Q}, 2^{AP}, \delta, \mathcal{Q}_0, F)$  non-blocking NBA  
representing the bad behaviors of an  $\omega$ -regular  
LT-property  $E$ , i.e.,  $\mathcal{L}_\omega(\mathcal{A}) = (2^{AP})^\omega \setminus E$

$\mathcal{T} = (\mathcal{S}, Act, \rightarrow, \mathcal{S}_0, AP, L)$  finite transition system  
without terminal states

$\mathcal{A} = (\mathcal{Q}, 2^{AP}, \delta, \mathcal{Q}_0, F)$  non-blocking NBA  
representing the bad behaviors of an  $\omega$ -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)  $Traces(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \emptyset$

$\mathcal{T} = (\mathcal{S}, \text{Act}, \rightarrow, \mathcal{S}_0, \text{AP}, L)$  finite transition system  
without terminal states

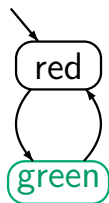
$\mathcal{A} = (\mathcal{Q}, 2^{\text{AP}}, \delta, \mathcal{Q}_0, F)$  non-blocking NBA  
representing the bad behaviors of an  $\omega$ -regular  
LT-property  $E$ , i.e.,  $\mathcal{L}_\omega(\mathcal{A}) = (2^{\text{AP}})^\omega \setminus E$

The following statements are equivalent:

- (1)  $\mathcal{T} \models E$
- (2)  $\text{Traces}(\mathcal{T}) \cap \mathcal{L}_\omega(\mathcal{A}) = \emptyset$
- (3)  $\mathcal{T} \otimes \mathcal{A} \models$  “eventually forever  $\neg F$ ”

# Example: $\omega$ -regular model checking

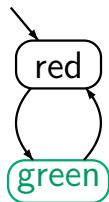
TS  $\mathcal{T}$



LT property: “infinitely often green”

# Example: $\omega$ -regular model checking

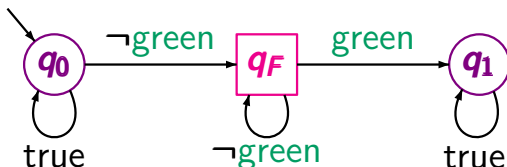
TS  $\mathcal{T}$



LT property: “infinitely often **green**”

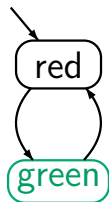
NBA  $\mathcal{A}$  for the complement

“from some moment on  $\neg$ **green**”



# Example: $\omega$ -regular model checking

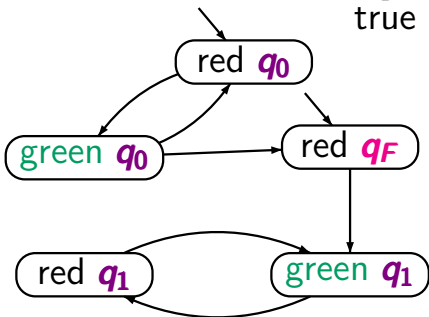
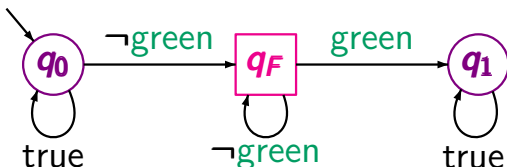
TS  $\mathcal{T}$



LT property: “infinitely often **green**”

NBA  $\mathcal{A}$  for the complement

“from some moment on  $\neg$ **green**”

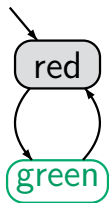


reachable fragment of the  
product TS  $\mathcal{T} \otimes \mathcal{A}$



# Example: $\omega$ -regular model checking

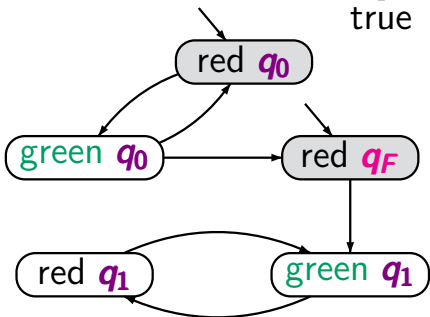
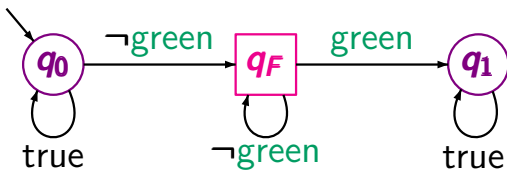
TS  $\mathcal{T}$



LT property: “infinitely often green”

NBA  $\mathcal{A}$  for the complement

“from some moment on  $\neg$ green”



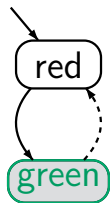
initial states:

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

$$\begin{aligned} q &\in \delta(q_0, L(\text{red})) \\ &= \delta(q_0, \emptyset) \\ &= \{q_0, q_F\} \end{aligned}$$

# Example: $\omega$ -regular model checking

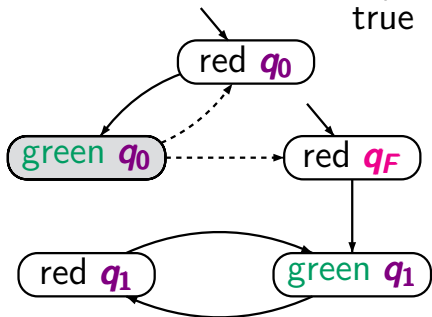
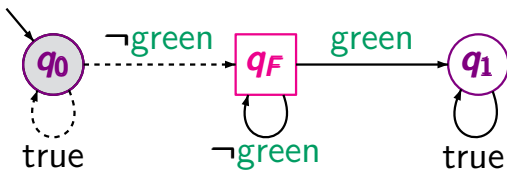
TS  $\mathcal{T}$



LT property: “infinitely often green”

NBA  $\mathcal{A}$  for the complement

“from some moment on  $\neg$ green”



transition

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

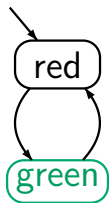
$q \in \delta(q_0, L(\text{red}))$

$= \delta(q_0, \emptyset)$

$= \{q_0, q_F\}$

# Example: $\omega$ -regular model checking

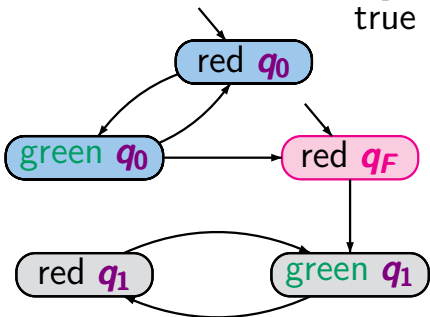
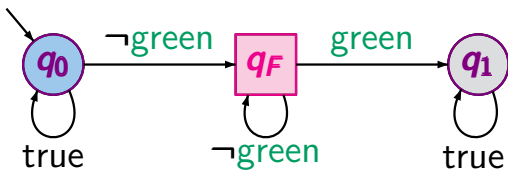
TS  $\mathcal{T}$



LT property: “infinitely often green”

NBA  $\mathcal{A}$  for the complement

“from some moment on  $\neg$ green”



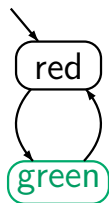
atomic propositions

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

obvious labeling function

# Example: $\omega$ -regular model checking

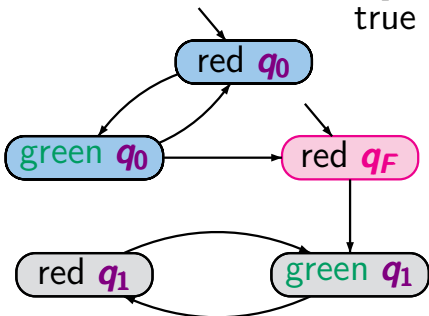
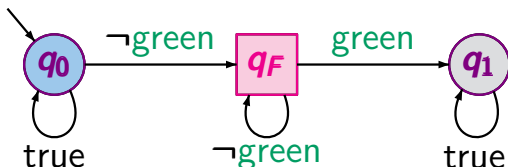
TS  $\mathcal{T}$



LT property: “infinitely often green”

NBA  $\mathcal{A}$  for the complement

“from some moment on  $\neg$ green”



atomic propositions

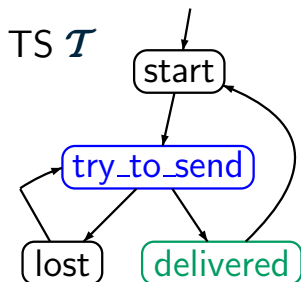
$$AP' = \{q_0, q_F, q_1\}$$

obvious labeling function

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

“eventually forever  $\neg F$ ”

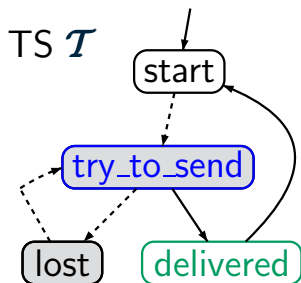
# Example: $\omega$ -regular model checking



$\omega$ -regular LT property  $E$ :

“each (repeatedly) sent message will eventually be delivered”

# Example: $\omega$ -regular model checking

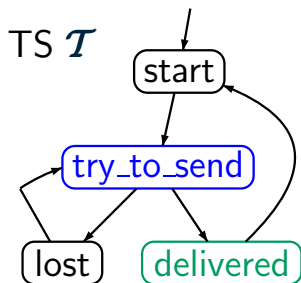


$\omega$ -regular LT property  $E$ :

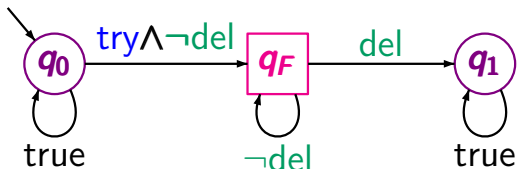
“each (repeatedly) sent message will eventually be delivered”

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

# Example: $\omega$ -regular model checking



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



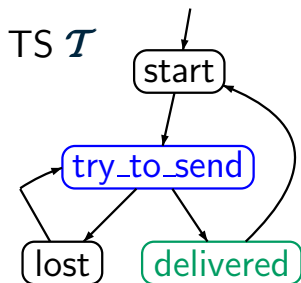
$\omega$ -regular LT property  $E$ :

“each (repeatedly) sent message will eventually be delivered”

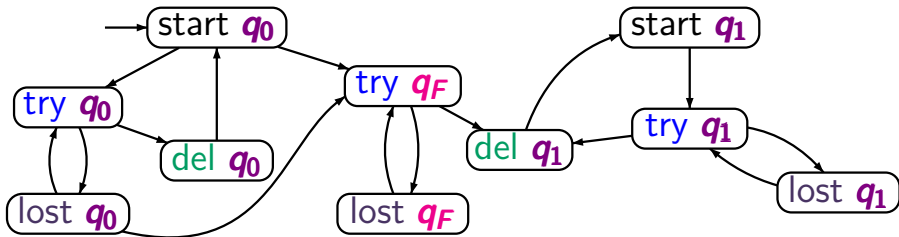
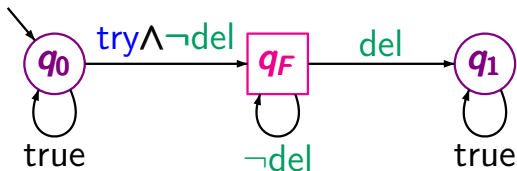
complement of  $E$ , i.e., LT property for the bad behaviors:

“never delivered after some trial”

# Example: $\omega$ -regular model checking



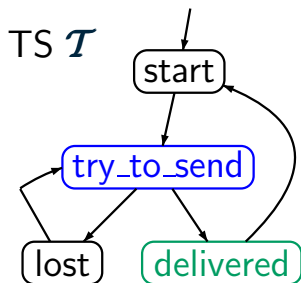
NBA  $\mathcal{A}$  for the bad behaviors



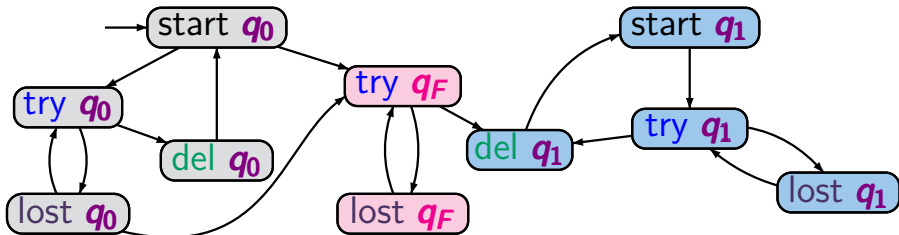
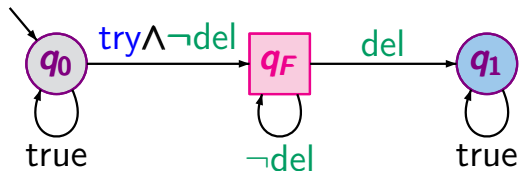
reachable fragment of the product-TS



# Example: $\omega$ -regular model checking

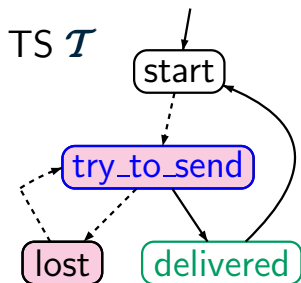


NBA  $\mathcal{A}$  for the bad behaviors

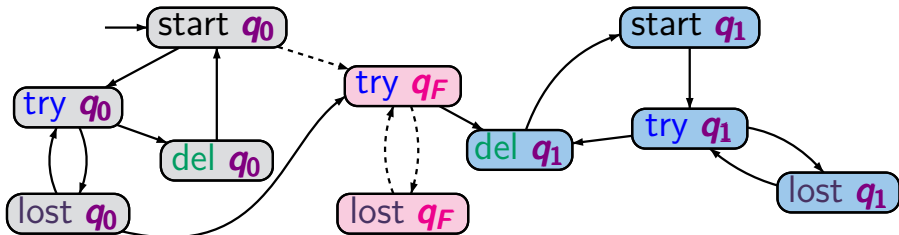
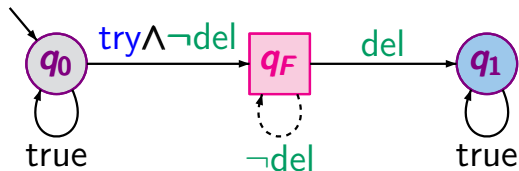


set of atomic propositions  $AP' = \{q_0, q_1, q_F\}$

# Example: $\omega$ -regular model checking



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



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



for regular safety property  $E$

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

iff  $Traces_{fin}(\mathcal{T}) \cap BadPref = \emptyset$

for regular safety property  $E$

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

iff  $Traces_{fin}(\mathcal{T}) \cap BadPref = \emptyset$

for  $\omega$ -regular property  $E$

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

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

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

for regular safety property  $E$

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

iff  $Traces_{fin}(\mathcal{T}) \cap \mathcal{L}(\mathcal{A}) = \emptyset$

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

for  $\omega$ -regular property  $E$

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

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

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

for regular safety property  $E$

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

iff  $Traces_{fin}(\mathcal{T}) \cap \mathcal{L}(\mathcal{A}) = \emptyset$

iff  $\mathcal{T} \otimes \mathcal{A} \models \text{“forever } \neg F\text{”}$

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

for  $\omega$ -regular property  $E$

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

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

iff  $\mathcal{T} \otimes \mathcal{A} \models \text{“eventually forever } \neg F\text{”}$

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

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

for regular safety property  $E$

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

iff  $Traces_{fin}(\mathcal{T}) \cap \mathcal{L}(\mathcal{A}) = \emptyset$

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

invariant  
checking

for  $\omega$ -regular property  $E$

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

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

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

persistence  
checking

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



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

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

*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  $s_0 s_1 s_2 s_3 \dots$  in  $\mathcal{T}$  s.t.  
 $s_i \not\models a$  for infinitely many  $i \geq 0$

*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  $s_0 s_1 s_2 s_3 \dots$  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  $s \dots s$

*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  $s_0 s_1 s_2 s_3 \dots$  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  $s \dots s$

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

*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  $C \cap \{s \in S : s \not\models a\} \neq \emptyset$   $\uparrow$

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

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

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”

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

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

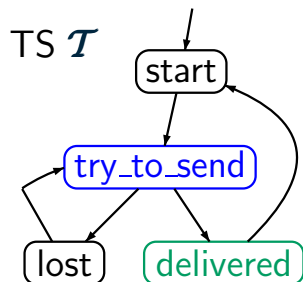
$\mathcal{T} \not\models$  “eventually forever  $a$ ”

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

*method:* calculate and analyze the **SCCs**

# Example: $\omega$ -regular model checking

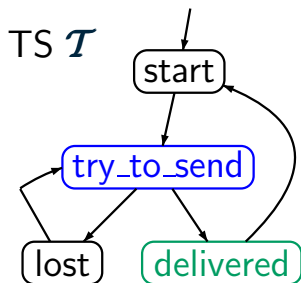


$\omega$ -regular LT property  $E$ :

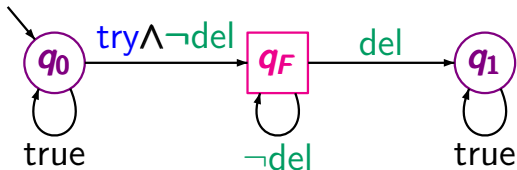
“each (repeatedly) sent message will eventually be delivered”



# Example: $\omega$ -regular model checking



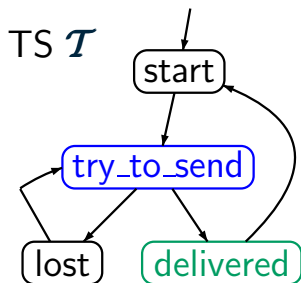
NBA  $\mathcal{A}$  for the bad behaviors



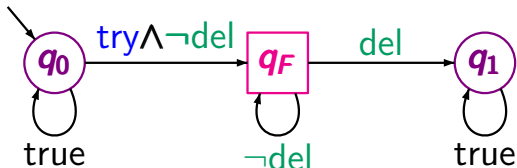
$\omega$ -regular LT property  $E$ :

“each (repeatedly) sent message will eventually be delivered”

# Example: $\omega$ -regular model checking



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



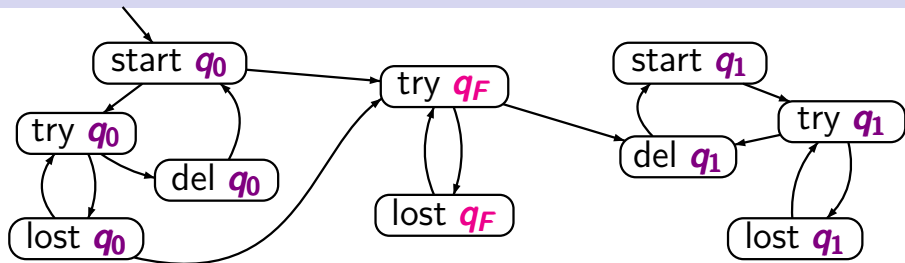
$\omega$ -regular LT property  $E$ :

“each (repeatedly) sent message will eventually be delivered”

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

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

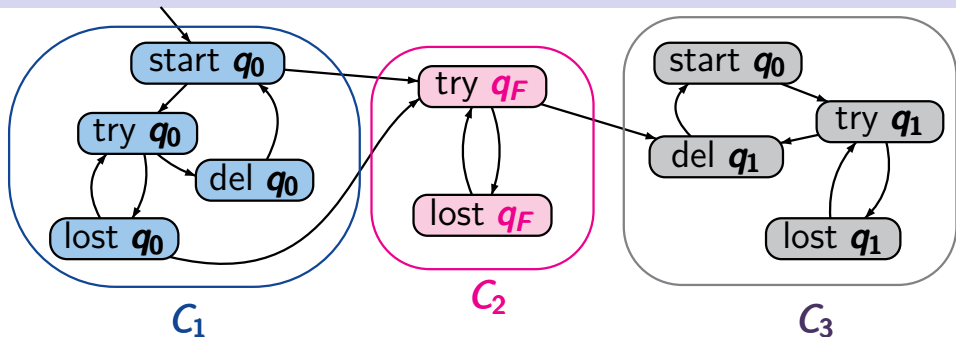
LTLMC3.2-12



persistence property: “eventually forever  $\neg q_F$ ”

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

LTLMC3.2-12

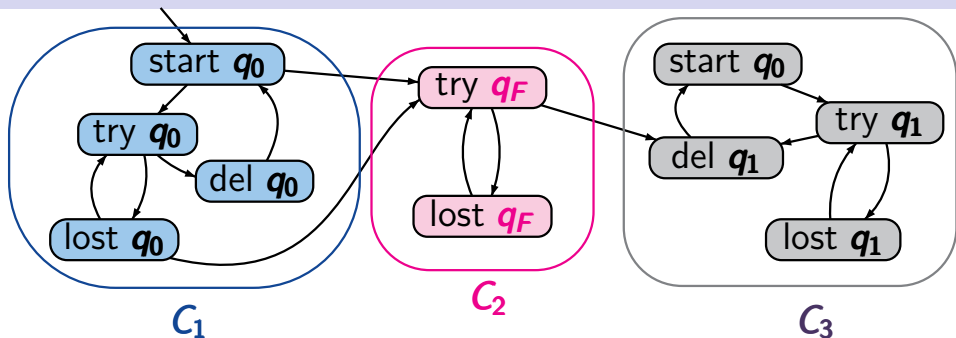


persistence property: “eventually forever  $\neg q_F$ ”

3 reachable SCCs:  $C_1$ ,  $C_2$ ,  $C_3$

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

LTLMC3.2-12



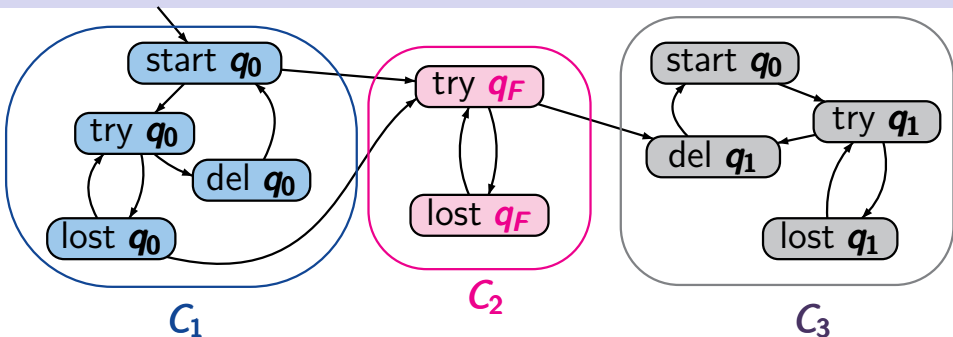
persistence property: “eventually forever  $\neg q_F$ ”

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

LTLMC3.2-12



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  $\neg q_F$ ”

$\mathcal{T} \not\models$  “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  $C \cap \{s \in S : s \not\models a\} \neq \emptyset$

$T \not\models$  “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  $C \cap \{s \in S : s \not\models a\} \neq \emptyset$

*method 1:* calculation and analysis of the SCCs



$T \not\models$  “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  $C \cap \{s \in S : s \not\models a\} \neq \emptyset$

*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

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

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

*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

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



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**.

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

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

The following statements are equivalent:

- (1)  $s$  belongs to a cycle  $s s_1 s_2 \dots s_k s$
- (2) The DFS started with  $s$  finds a backward edge  $s' \rightarrow s$ .

*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  $s' \rightarrow s$

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

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

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



*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”

*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

LTLMC3.2-14

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”

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

FI

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

# Persistence checking ← Nested DFS

LTLMC3.2-14

*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  $s \neq a$ ;  
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  $s \neq a$  are marked;  
return “yes”



# Time complexity of nested DFS

LTLMC3.2-14

REPEAT

1. DFS: visits all reachable states

choose an unmarked **reachable** state  $s$  with  $s \neq a$ ;  
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  $s \neq a$  are marked;  
return "yes"

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

# Time complexity of nested DFS

LTLMC3.2-14

REPEAT

1. DFS: visits all reachable states

choose an unmarked **reachable** state  $s$  with  $s \neq a$ ;  
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  $s \neq a$  are marked;  
return "yes"

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

cost of **CYCLE\_CHECK**( $s$ )  
caused by each state  $s \neq a$

# Time complexity of nested DFS

LTLMC3.2-14

REPEAT

1. DFS: visits all reachable states

choose an unmarked **reachable** state  $s$  with  $s \neq a$ ;  
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  $s \neq a$  are marked;  
return "yes"

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

$\Theta(|S|)$  states  
with  $s \neq a$

cost of **CYCLE\_CHECK**( $s$ )  
caused by each state  $s \neq a$



# Time complexity of nested DFS

LTLMC3.2-14

REPEAT

1. DFS: visits all reachable states

choose an unmarked **reachable** state  $s$  with  $s \neq a$ ;  
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  $s \neq a$  are marked;  
return "yes"

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

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

- 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  $s' \rightarrow s$

- 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*

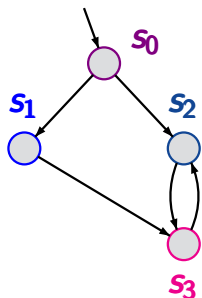
- serves to check whether  $\mathcal{T} \models$  “eventually forever  $a$ ”
- relies on two DFS running in an interleaved way

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



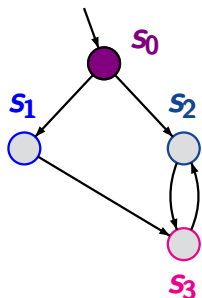
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

# Example: nested DFS

LTLMC3.2-15



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

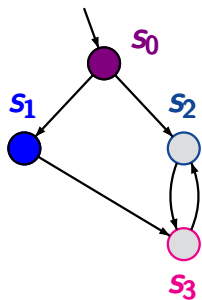
$DFS(s_0)$

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



# Example: nested DFS

LTLMC3.2-15



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

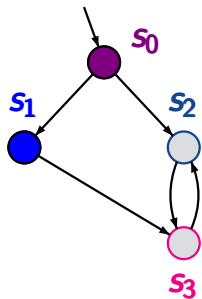
$DFS(s_0)$

$DFS(s_1)$

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

# Example: nested DFS

LTLMC3.2-15



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

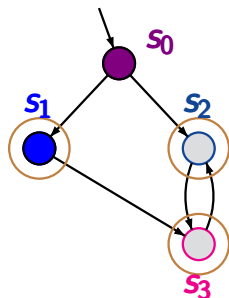
$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

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

# Example: nested DFS

LTLMC3.2-15



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

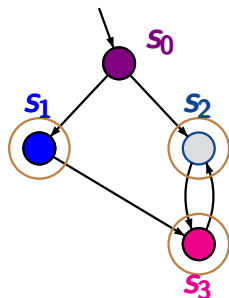
$CYCLE\_CHECK(s_1)$

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

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

# Example: nested DFS

LTLMC3.2-15



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

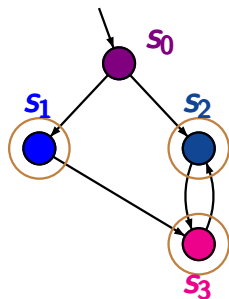
$V = \{s_1, s_2, s_3\}$

$DFS(s_3)$

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

# Example: nested DFS

LTLMC3.2-15



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

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

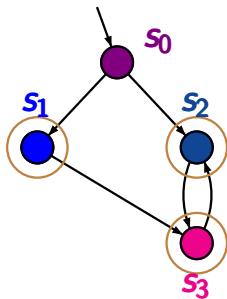
$DFS(s_3)$

$DFS(s_2)$

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

# Example: nested DFS

LTLMC3.2-15



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

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

$DFS(s_3)$

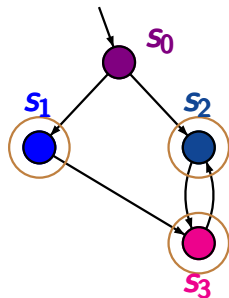
$DFS(s_2)$

$CYCLE\_CHECK(s_2)$

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

# Example: nested DFS ← fails

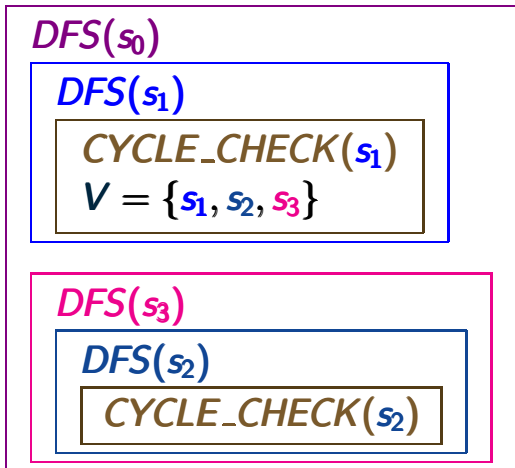
LTLMC3.2-15



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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



returns wrong  
answer “yes”

- 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  $s \not\models a$



- 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  $s \not\models a$

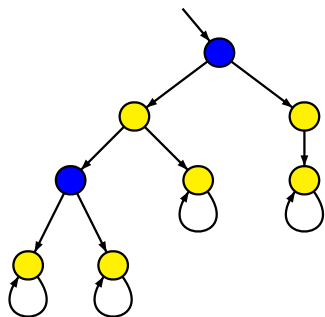
- checks whether  $s$  belongs to a cycle by searching a backward edge  $s' \rightarrow s$
- ignores states that have been visited in previous calls of *CYCLE\_CHECK* by using a global visiting set  $V$  for the 2. **DFS**

- 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  $s \neq a$

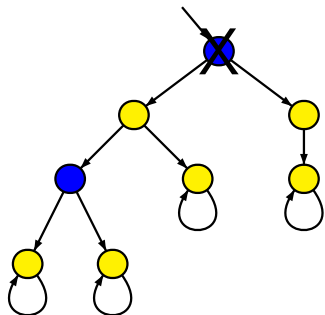
- checks whether  $s$  belongs to a cycle by searching a backward edge  $s' \rightarrow s$
- ignores states that have been visited in previous calls of *CYCLE\_CHECK* by using a global visiting set  $V$  for the 2. **DFS**
- is called for state  $s$  after  $s$  is **fully expanded** in the 1. **DFS**



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

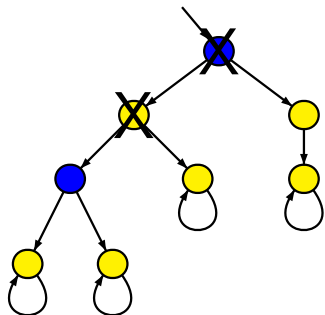
1. **DFS:** visits all reachable states

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



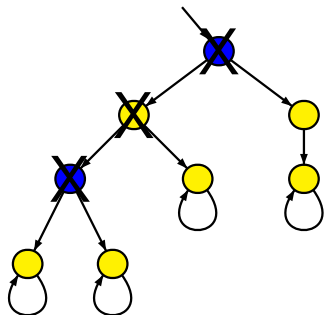
$\mathcal{T} \models$  “eventually forever  $\neg$ *blue*”

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  $s' \rightarrow s$



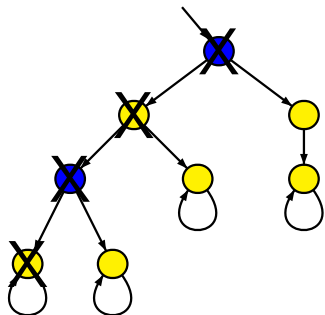
$\mathcal{T} \models$  “eventually forever  $\neg$ *blue*”

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  $s' \rightarrow s$



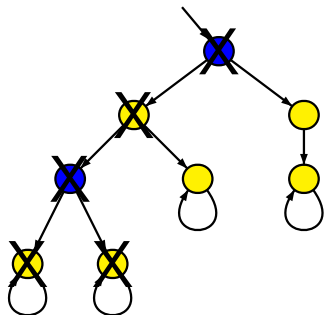
$\mathcal{T} \models$  “eventually forever  $\neg$ *blue*”

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  $s' \rightarrow s$



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

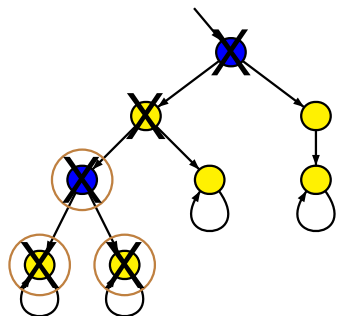
1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE\_CHECK*( $s$ ) for  $s \models$  *blue*  
 checks whether  $s$  belongs to a cycle  
 by searching a backward-edge  $s' \rightarrow s$



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

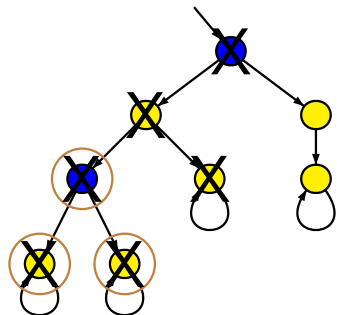
1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE\_CHECK*( $s$ ) for  $s \models$  *blue*  
 checks whether  $s$  belongs to a cycle  
 by searching a backward-edge  $s' \rightarrow s$





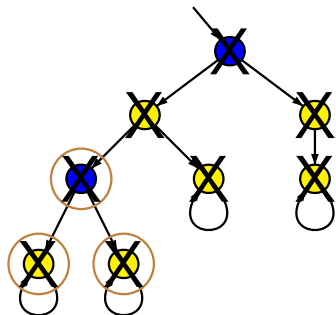
$\mathcal{T} \models$  “eventually forever  $\neg$ *blue*”

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  $s' \rightarrow s$



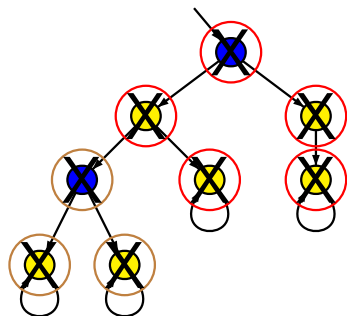
$\mathcal{T} \models$  “eventually forever  $\neg$ *blue*”

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  $s' \rightarrow s$



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



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

$U := \emptyset;$

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

## Nested DFS (pseudo code)

LTLMC3.2-17

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

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

# Nested DFS (pseudo code)

LTLMC3.2-17

$U := \emptyset;$   $\leftarrow$  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)

LTLMC3.2-17

$U := \emptyset$ ;  $\leftarrow$  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)

LTLMC3.2-17

$U := \emptyset$ ;  $\leftarrow$  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 ;

IF  $s \neq a$  THEN

IF  $CYCLE\_CHECK(s)$

FI

pseudo code for  
 $DFS(s)$

# Nested DFS (pseudo code)

LTLMC3.2-17

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

$V := \emptyset$   $\leftarrow$  global visiting set of 2. 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 ;

IF  $s \neq a$  THEN

IF  $CYCLE\_CHECK(s)$

FI

pseudo code for  
 $DFS(s)$

# Nested DFS (pseudo code)

LTLMC3.2-17

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

$V := \emptyset$   $\leftarrow$  global visiting set of 2. 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 ;

IF  $s \neq a$  THEN

IF  $CYCLE\_CHECK(s)$  THEN return "no" FI

FI

pseudo code for  
 $DFS(s)$

# Nested DFS (pseudo code)

LTLMC3.2-17

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

$V := \emptyset$   $\leftarrow$  global visiting set of 2. 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 ;

IF  $s \neq a$  THEN

IF  $CYCLE\_CHECK(s)$  THEN return "no" FI

FI

$T \neq$  "eventually forever  $a$ "

pseudo code for  
 $DFS(s)$

# Nested DFS (pseudo code)

LTLMC3.2-17

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

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

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

return “yes”  $\leftarrow T \models$  “eventually forever  $a$ ”

IF  $s \notin U$  THEN

insert  $s$  in  $U$ ;

pseudo code for  
 $DFS(s)$

FOR ALL  $s' \in Post(s)$  DO  $DFS(s')$  OD ;

IF  $s \not\models a$  THEN

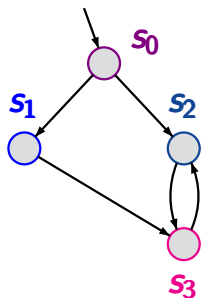
IF  $CYCLE\_CHECK(s)$  THEN return “no” FI

FI

FI

# Example: nested DFS

LTLMC3.2-33



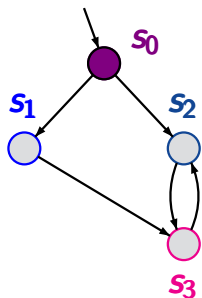
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

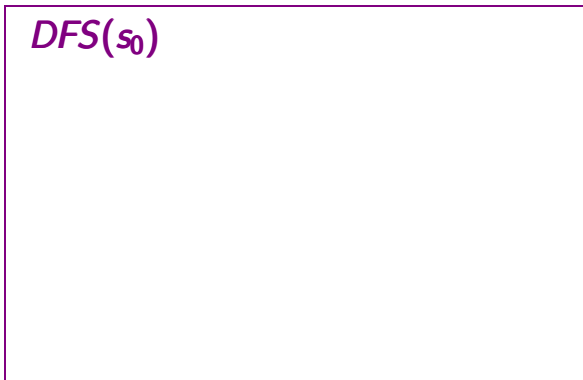
# Example: nested DFS

LTLMC3.2-33



$s_1, s_2 \not\models a$

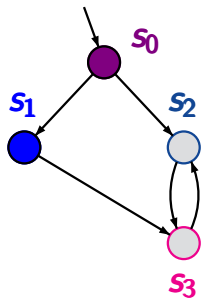
$s_0, s_3 \models a$



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

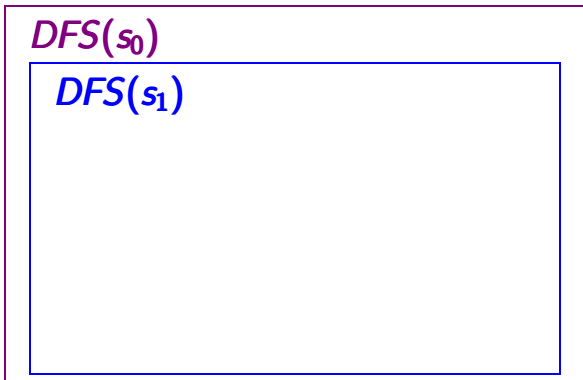
# Example: nested DFS

LTLMC3.2-33



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

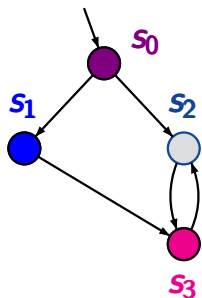


$\mathcal{T} \not\models$  “eventually forever  $a$ ”



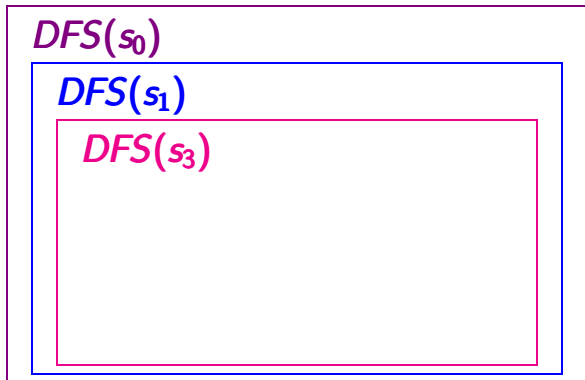
# Example: nested DFS

LTLMC3.2-33



$s_1, s_2 \not\models a$

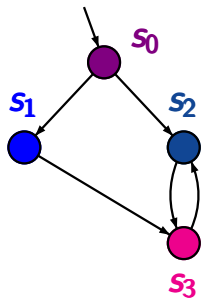
$s_0, s_3 \models a$



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

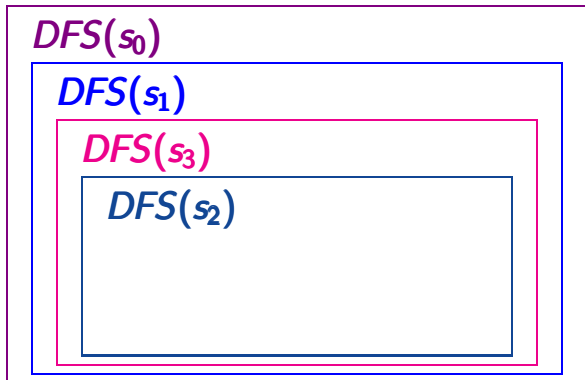
# Example: nested DFS

LTLMC3.2-33



$s_1, s_2 \not\models a$

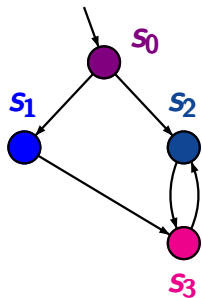
$s_0, s_3 \models a$



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

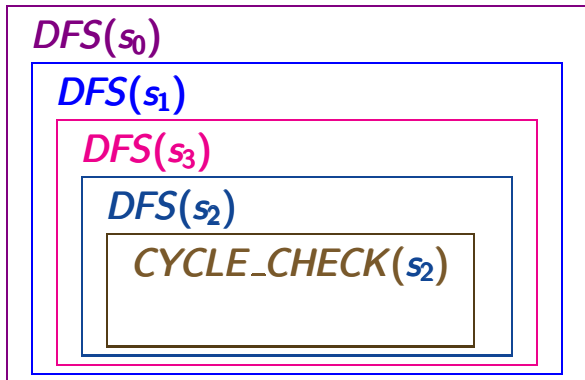
# Example: nested DFS

LTLMC3.2-33



$s_1, s_2 \not\models a$

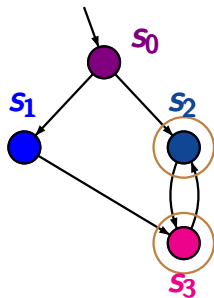
$s_0, s_3 \models a$



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

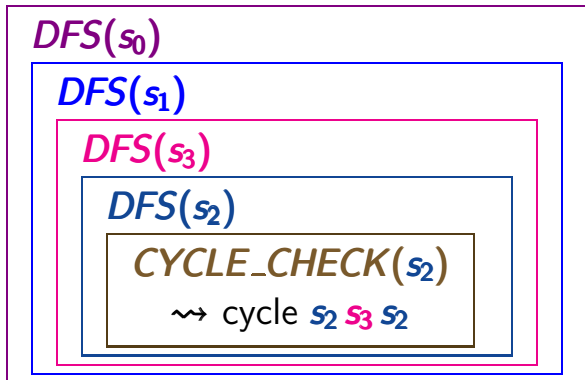
# Example: nested DFS

LTLMC3.2-33



$s_1, s_2 \not\models a$

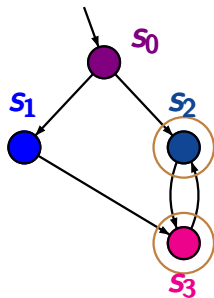
$s_0, s_3 \models a$



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

# Example: nested DFS

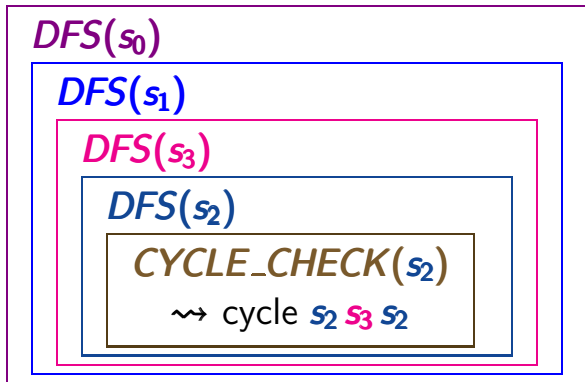
LTLMC3.2-33



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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



↑  
returns correct  
answer “no”



- 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

LTLMC3.2-34

*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

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

where  $s_n \not\models a$



*input:* finite TS  $\mathcal{T} = (\mathcal{S}, \text{Act}, \rightarrow, \mathcal{S}_0, \text{AP}, L)$   
persistence condition  $a \in \text{AP}$

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



initial path fragment of the form

$\mathcal{S}_0 \dots \mathcal{S}_{n-1} \mathcal{S}_n \mathcal{S}_{n+1} \dots \mathcal{S}_{n+m-1} \mathcal{S}_n$

where  $\mathcal{S}_n \not\models a$

... iterative formulation with **2 stacks** ...

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

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

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

$V := \emptyset; \xi := \emptyset;$  ← 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  $s_0 \notin U$  DO

OD

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

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

OD

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

    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



$U := \emptyset; \pi := \emptyset; \leftarrow$  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

$s := Top(\pi)$ ;

IF  $Post(s) \not\subseteq U$

OD OD FI

$U := \emptyset; \pi := \emptyset; \leftarrow$  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

$s := Top(\pi)$ ;

IF  $Post(s) \not\subseteq U$

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

insert  $s'$  in  $U$ ;  $Push(\pi, s')$

OD OD FI

$U := \emptyset; \pi := \emptyset; \leftarrow$  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

$s := Top(\pi)$ ;

        IF  $Post(s) \not\subseteq U$

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

            insert  $s'$  in  $U$ ;  $Push(\pi, s')$

        ELSE  $Pop(\pi)$ ;

OD OD FI

$U := \emptyset; \pi := \emptyset; \leftarrow$  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

$s := Top(\pi)$ ;

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  $s \neq a$  and  $CYCLE\_CHECK(s)$

THEN return "no"

OD OD

FI

FI

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

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

WHILE  $S_0 \notin U$  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  $s \neq a$  and  $CYCLE\_CHECK(s)$

THEN return "no" +  $reverse(\pi, \xi)$  FI

OD OD FI

$U := \emptyset; \pi := \emptyset; \leftarrow$  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

$s := Top(\pi)$ ;

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  $s \neq a$  and  $CYCLE\_CHECK(s)$

THEN return "no" +  $reverse(\pi, \xi)$  FI

OD OD FI

return "yes"

- is called for  $s \neq a$
- checks whether  $s$  belongs to a cycle
- uses global visiting set  $V$  and stack  $\xi$

## Algorithm *CYCLE\_CHECK*(*s*)

LTLMC3.2-35

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



## Algorithm *CYCLE\_CHECK*(*s*)

LTLMC3.2-35

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

WHILE  $\xi \neq \emptyset$  DO

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

WHILE  $\xi \neq \emptyset$  DO

$s' := \text{Top}(\xi)$ ;

    IF  $s \in \text{Post}(s')$

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

WHILE  $\xi \neq \emptyset$  DO

$s' := \text{Top}(\xi)$ ;

    IF  $s \in \text{Post}(s')$

        THEN *Push*( $\xi$ , *s*); return “true”

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

WHILE  $\xi \neq \emptyset$  DO

$s' := \text{Top}(\xi)$ ;

    IF  $s \in \text{Post}(s')$

        THEN *Push*( $\xi, s$ ); return “true”

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

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

WHILE  $\xi \neq \emptyset$  DO

$s' := \text{Top}(\xi)$ ;

    IF  $s \in \text{Post}(s')$

        THEN *Push*( $\xi, s$ ); return “true”

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

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

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

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

WHILE  $\xi \neq \emptyset$  DO

$s' := \text{Top}(\xi)$ ;

    IF  $s \in \text{Post}(s')$

        THEN *Push*( $\xi, s$ ); return “true”

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

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

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

    ELSE *Pop*( $\xi$ )

FI

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

WHILE  $\xi \neq \emptyset$  DO

$s' := \text{Top}(\xi)$ ;

    IF  $s \in \text{Post}(s')$

        THEN *Push*( $\xi, s$ ); return “true”

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

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

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

    ELSE *Pop*( $\xi$ )

    FI

FI

OD

return “false”

# Algorithm *CYCLE\_CHECK*(*s*)

LTLMC3.2-35B

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

WHILE  $\xi \neq \emptyset$  DO

$s' := \text{Top}(\xi)$ ;

    IF  $s \in \text{Post}(s')$

        THEN *Push*( $\xi, s$ ); return “true”

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

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

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

    ELSE *Pop*( $\xi$ )

    FI

OD FI

return “false”



# Algorithm *CYCLE\_CHECK*(*s*)

LTLMC3.2-35B

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

WHILE  $\xi \neq \emptyset$  DO

$s' := \text{Top}(\xi)$ ;

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

THEN *Push*( $\xi, s$ ); return “true”

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

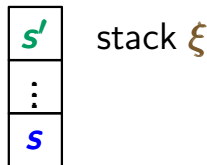
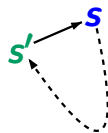
THEN choose  $s'' \in \text{Post}(s') \setminus V$ ;  
insert  $s''$  in  $V$ ; *Push*( $\xi, s''$ );

ELSE *Pop*( $\xi$ )

FI

OD FI

return “false”



# Algorithm *CYCLE\_CHECK*( $s$ )

LTLMC3.2-35B

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

WHILE  $\xi \neq \emptyset$  DO

$s' := \text{Top}(\xi)$ ;

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

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

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

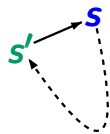
THEN choose  $s'' \in \text{Post}(s') \setminus V$ ;  
insert  $s''$  in  $V$ ; *Push*( $\xi, s''$ );

ELSE *Pop*( $\xi$ )

FI

OD FI

return "false"



stack  $\xi$

# Nested DFS with counterexample generation

LTLMC3.2-35C

```
 $U := \emptyset; \pi := \emptyset; V := \emptyset; \xi := \emptyset;$   
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  $s \not\equiv a$  and  $CYCLE\_CHECK(s)$   
          THEN return "no" +  $reverse(\pi, \xi)$  FI  
    OD  
  FI  
OD  
return "yes"
```

# Nested DFS with counterexample generation

LTLMC3.2-35D

$U := \emptyset; \pi := \emptyset; V := \emptyset; \xi := \emptyset;$

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  $s \neq a$  and  $CYCLE\_CHECK(s)$

THEN return "no" +  $reverse(\pi, \xi)$  FI

OD OD FI

return "yes"

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

←  $DFS(s)$  ends

outer DFS: visits all reachable states  $s$

inner DFS: algorithm *CYCLE\_CHECK*( $s$ )

- is called for  $s \neq a$  when *DFS*( $s$ ) is finished
- uses global data structures  $V$  and  $\xi$

outer DFS: visits all reachable states  $s$

inner DFS: algorithm  $CYCLE\_CHECK(s)$

- is called for  $s \neq a$  when  $DFS(s)$  is finished
- uses global data structures  $V$  and  $\xi$

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

$\xi$ : stack for counterexample

outer DFS: visits all reachable states  $s$

inner DFS: algorithm  $CYCLE\_CHECK(s)$

- is called for  $s \neq a$  when  $DFS(s)$  is finished
- uses global data structures  $V$  and  $\xi$

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

$\xi$ : stack for counterexample

**soundness:** 1. termination  
2. partial correctness

outer DFS: visits all reachable states  $s$

inner DFS: algorithm  $CYCLE\_CHECK(s)$

- is called for  $s \neq a$  when  $DFS(s)$  is finished
- uses global data structures  $V$  and  $\xi$

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

$\xi$ : stack for counterexample

**soundness:** 1. termination ✓  
2. partial correctness



outer DFS: visits all reachable states  $s$

inner DFS: algorithm  $CYCLE\_CHECK(s)$

- is called for  $s \neq a$  when  $DFS(s)$  is finished
- uses global data structures  $V$  and  $\xi$

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

$\xi$ : stack for counterexample

$\mathcal{T} \models$  “eventually forever  $a$ ”  
iff the nested DFS returns “yes”

outer DFS: visits all reachable states  $s$

inner DFS: algorithm  $CYCLE\_CHECK(s)$

- is called for  $s \neq a$  when  $DFS(s)$  is finished
- uses global data structures  $V$  and  $\xi$

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

$\xi$ : stack for counterexample

$T \neq$  “eventually forever  $a$ ”  
iff the nested DFS returns “no”

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

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

*Proof of “ $\Leftarrow$ ”:*

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

*Proof of “ $\Leftarrow$ ”:*

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  $t \rightarrow s$ .

$T \not\models$  “eventually forever  $a$ ”  
iff the nested DFS returns “no”

*Proof of “ $\Leftarrow$ ”:*

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

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

*Proof of “ $\Leftarrow$ ”:*

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

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

*Proof of “ $\Leftarrow$ ”:*

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  $\pi = s_0 \dots s_{n-1} (s t_1 \dots t_k)^\omega$  in  $\mathcal{T}$  s.t.  $trace(\pi) \notin$  “eventually forever  $a$ ”



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

*Proof of “ $\Leftarrow$ ”:*

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  $\pi = s_0 \dots s_{n-1} (s t_1 \dots t_k)^\omega$  in  $\mathcal{T}$  s.t.  $trace(\pi) \notin$  “eventually forever  $a$ ”

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

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

*Proof of “ $\implies$ ”:*

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

*Proof of “ $\implies$ ”:* show that:

When  $CYCLE\_CHECK(s)$  is called then there is  
no cycle  $t_0 t_1 \dots t_k$  in  $\mathcal{T}$  s.t.  $s = t_0 = t_k$  and


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

global visiting set of the inner DFS

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

*Proof* of “ $\implies$ ”: show that:

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

  
global visiting set of the inner DFS

Hence: if  $s$  belongs to a cycle then  $CYCLE\_CHECK(s)$   
will find a backward edge  $t \rightarrow s$



# Further improvements of the nested DFS

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

$V := \emptyset; \xi := \emptyset;$

WHILE  $S_0 \not\subseteq U$  DO

    choose  $s_0 \in S_0 \setminus U;$

$\vdots$

    WHILE  $\pi \neq \emptyset$  DO

$s := \text{Top}(\pi);$

        IF  $\text{Post}(s) \not\subseteq U$

            THEN ...

            ELSE  $\text{Pop}(\pi);$

            IF  $s \neq a$  and  $\text{CYCLE\_CHECK}(s)$  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  $s_0 \in S_0 \setminus U;$

$\vdots$

    WHILE  $\pi \neq \emptyset$  DO

$s := Top(\pi);$

        IF  $Post(s) \not\subseteq U$

            THEN ...

            ELSE  $Pop(\pi);$

            IF  $s \neq a$  and  $CYCLE\_CHECK(s)$  THEN ...

    FI

OD

OD

on the fly construction

# Further improvements of the nested DFS

LTLMC3.2-37

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

$V := \emptyset; \xi := \emptyset;$

WHILE  $S_0 \not\subseteq U$  DO

    choose  $s_0 \in S_0 \setminus U;$

$\vdots$

    WHILE  $\pi \neq \emptyset$  DO

$s := Top(\pi);$

        IF  $Post(s) \not\subseteq U$

            THEN ...

            ELSE  $Pop(\pi);$

            IF  $s \neq a$  and  $CYCLE\_CHECK(s)$  THEN ...

    FI

OD

OD

on the fly construction  
hash techniques for  $U$  and  $V$



# Further improvements of the nested DFS

LTLMC3.2-37

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

$V := \emptyset; \xi := \emptyset;$

WHILE  $S_0 \not\subseteq U$  DO

    choose  $s_0 \in S_0 \setminus U;$

$\vdots$

    WHILE  $\pi \neq \emptyset$  DO

$s := Top(\pi);$

        IF  $Post(s) \not\subseteq U$

            THEN ...

            ELSE  $Pop(\pi);$

            IF  $s \neq a$  and  $CYCLE\_CHECK(s)$  THEN ...

    FI

OD

OD

on the fly construction

hash techniques for  $U$  and  $V$

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

# Further improvements of the nested DFS

LTLMC3.2-37

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

$V := \emptyset; \xi := \emptyset;$

WHILE  $S_0 \not\subseteq U$  DO

    choose  $s_0 \in S_0 \setminus U;$

$\vdots$

    WHILE  $\pi \neq \emptyset$  DO

$s := Top(\pi);$

        IF  $Post(s) \not\subseteq U$

            THEN ...

            ELSE  $Pop(\pi);$

            IF  $s \neq a$  and  $CYCLE\_CHECK(s)$  THEN ...

    FI

OD

OD

on the fly construction

hash techniques for  $U$  and  $V$

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

early termination of

$CYCLE\_CHECK$ , e.g.,

if a state in  $\pi$  is visited