Persistence checking

given:	finite transition system $\mathcal{T}$ over AP persistence condition $a \in AP$
question:	does $\mathcal{T} \models$ "eventually forever a" hold ?

Persistence checking

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

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

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

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

Persistence checking

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

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

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

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

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

Persistence checking

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

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

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

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

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

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

Persistence checking

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

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

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

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

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

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

Persistence checking

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

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

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

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

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

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

Persistence checking

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

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

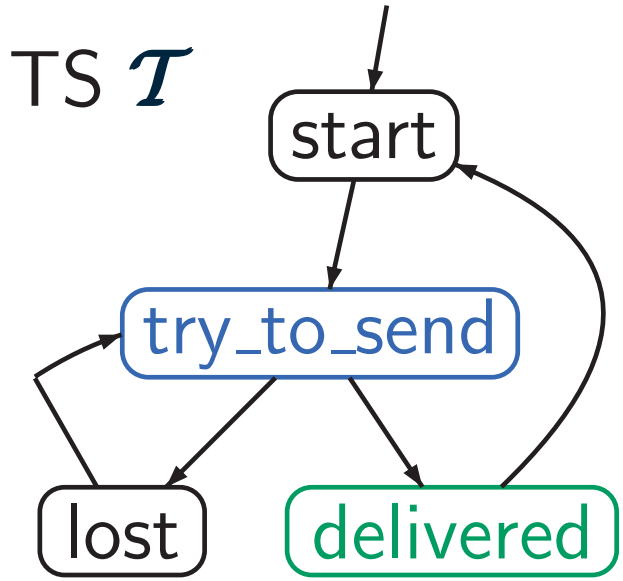
$\mathcal{T} \not\models$ "eventually forever a"

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

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

method: calculate and analyze the **SCCs**

Example: ω-regular model checking

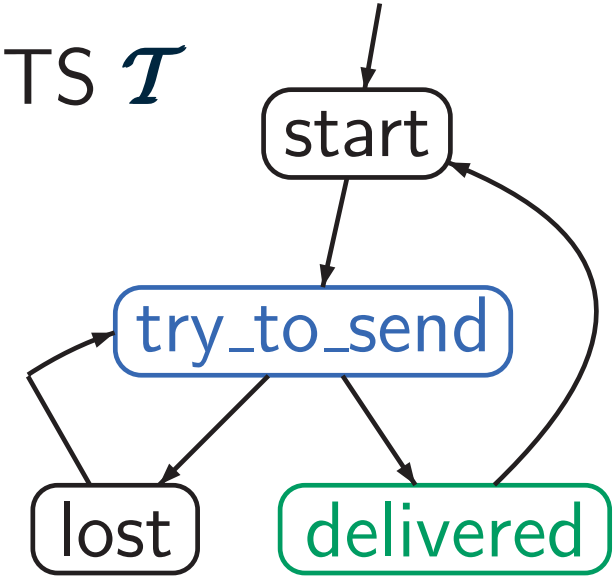


ω-regular LT property E:

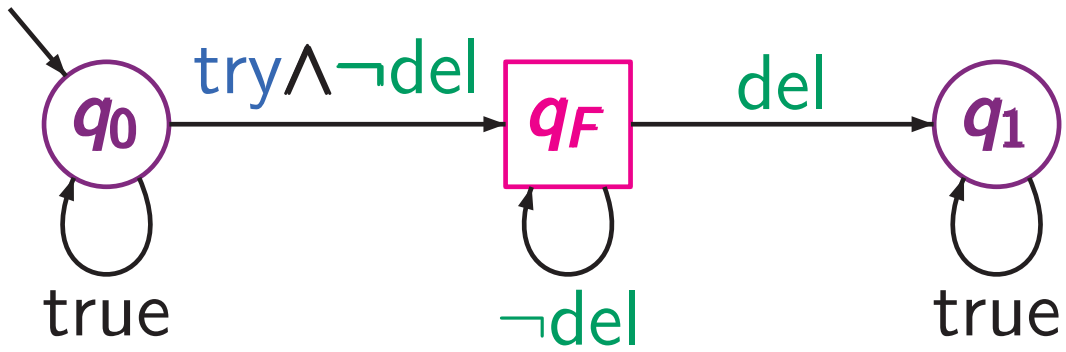
"each (repeatedly) sent message will eventually be delivered"

Example: ω-regular model checking

TS $\mathcal{T}$



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

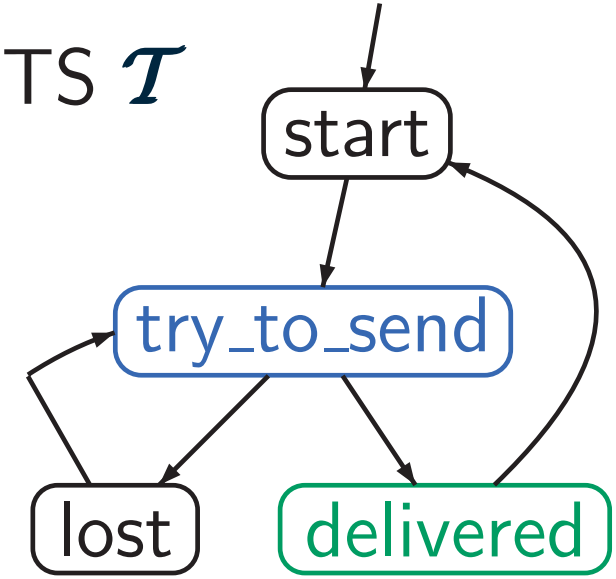


ω-regular LT property E:

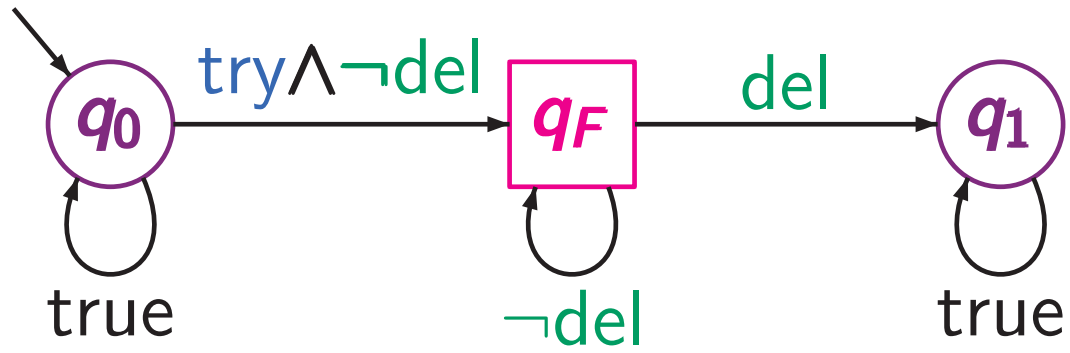
"each (repeatedly) sent message will eventually be delivered"

Example: ω-regular model checking

TS $\mathcal{T}$



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

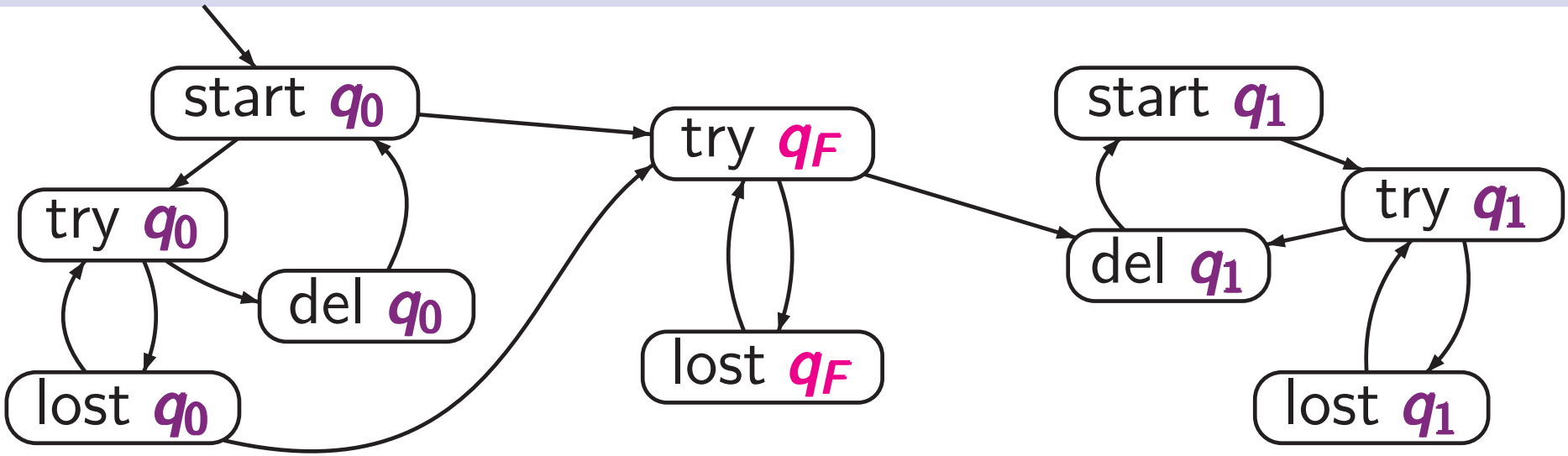


ω-regular LT property E:

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

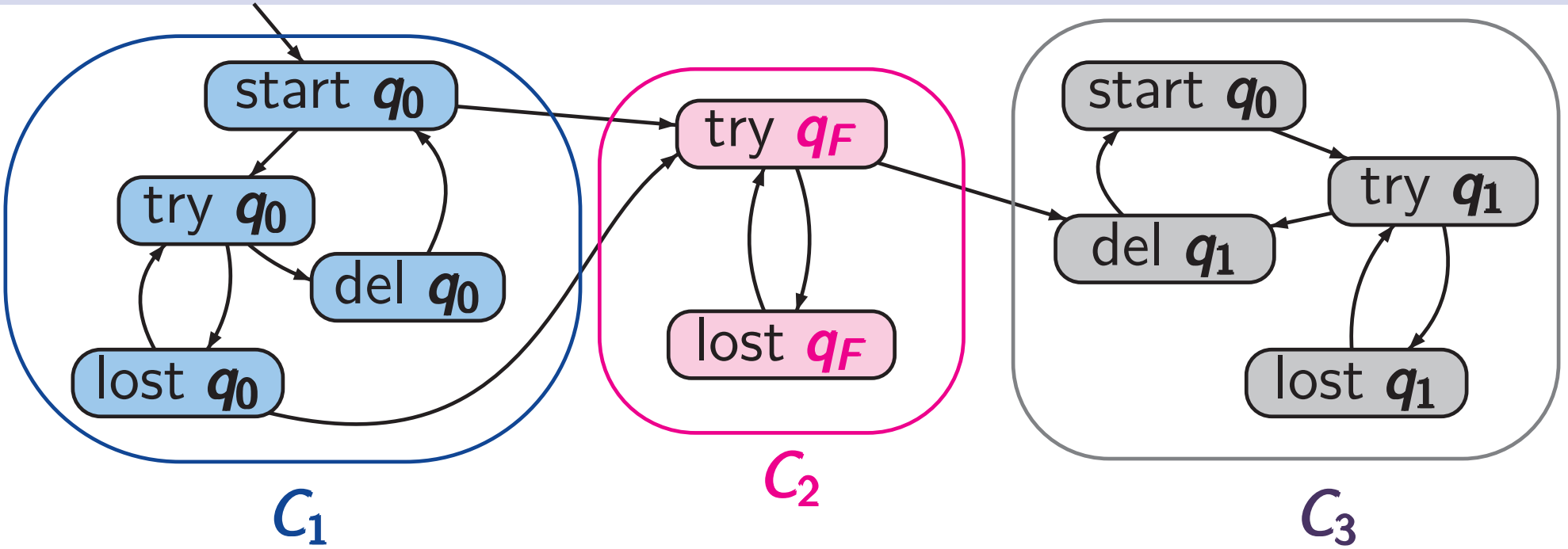
... analysis of the **SCCs** in product $\mathcal{T} \otimes \mathcal{A}$...

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



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

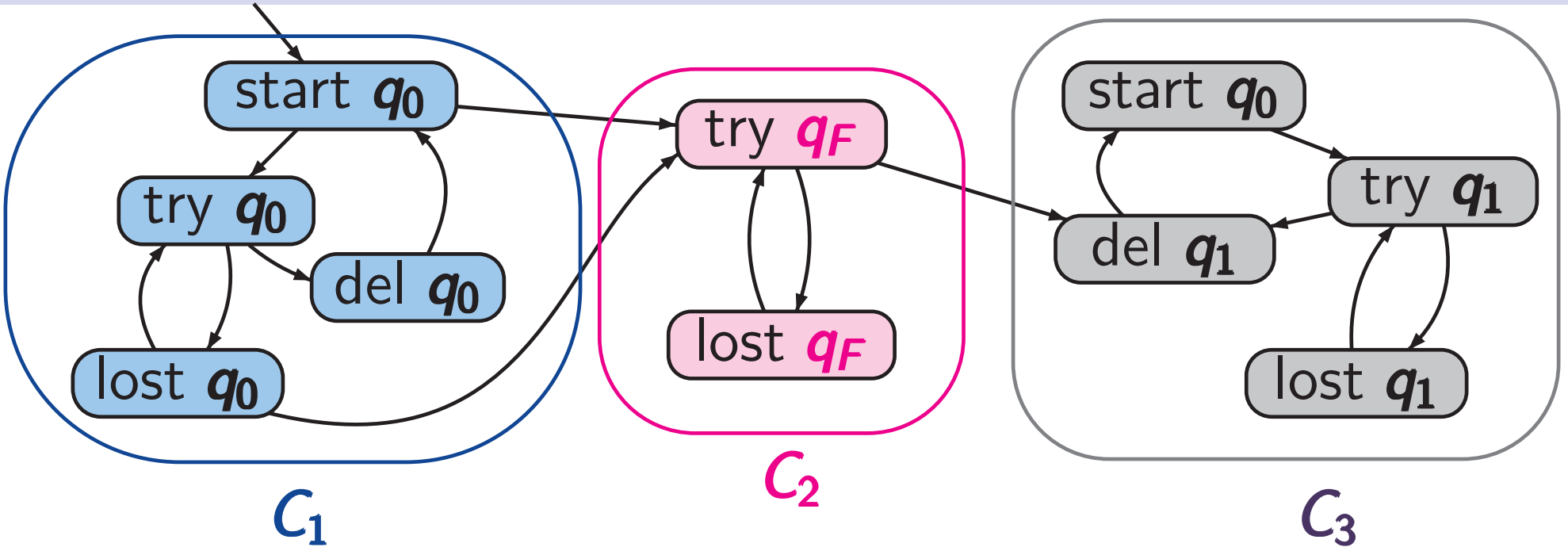
Example: persistence checking $\mathcal{T} \otimes \mathcal{A}$



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

3 reachable SCCs: C_1, C_2, C_3

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

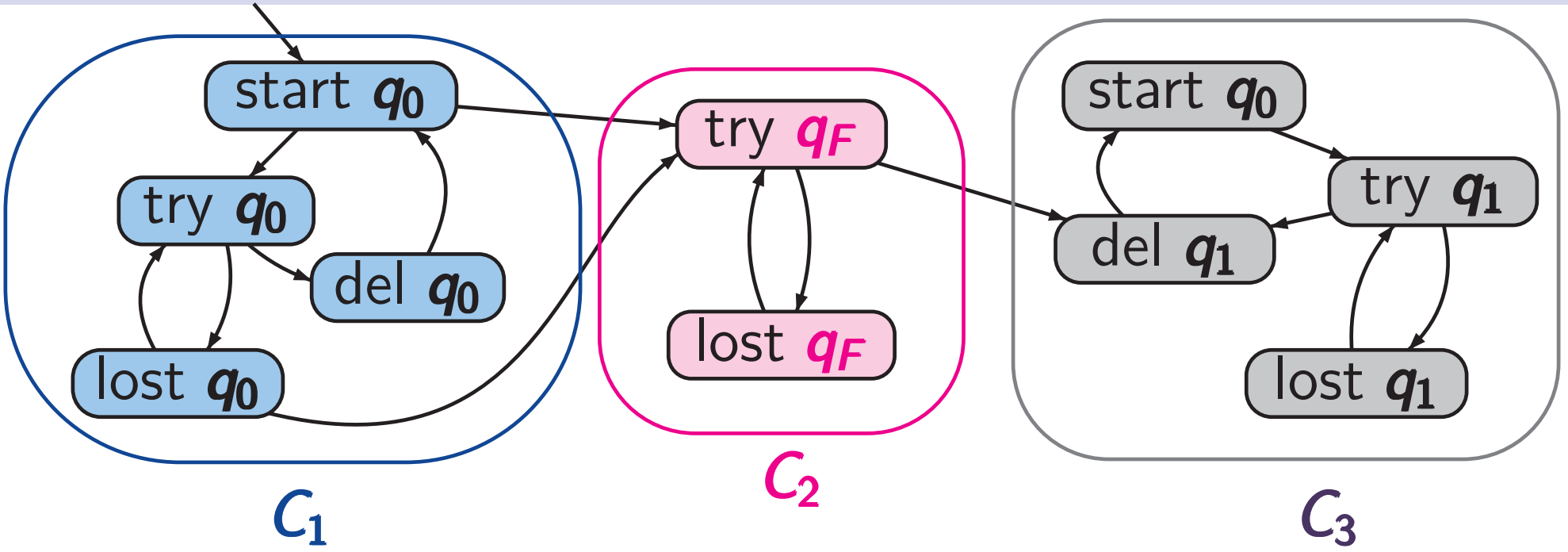


persistence property: "eventually forever $\neg q_F$"

3 reachable SCCs: C_1, C_2, C_3

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

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



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

3 reachable SCCs: C_1, C_2, C_3

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

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

Persistence checking algorithms based on ...

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

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

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

Persistence checking algorithms based on ...

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

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

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

method 1: calculation and analysis of the SCCs

Persistence checking algorithms based on ...

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

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

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

method 1: calculation and analysis of the SCCs

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

Persistence checking algorithms based on ...

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

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

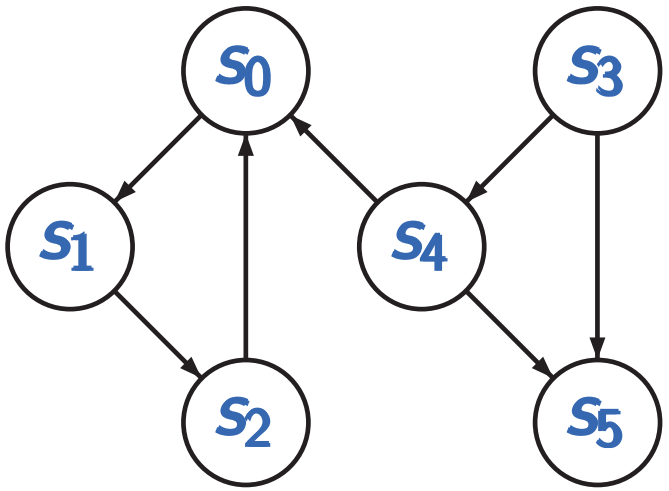
iff there exists a non-trivial reachable SCC C with $C \cap \{s \in S : s \not\models a\} \neq \varnothing$

method 1: calculation and analysis of the SCCs

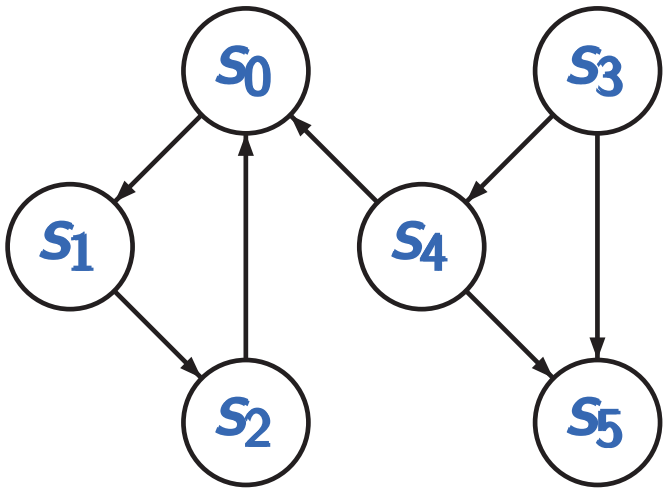
- algorithm to compute the SCCs rely on an exploration of the full (reachable) state space
- not adequate for on-the-fly analysis

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

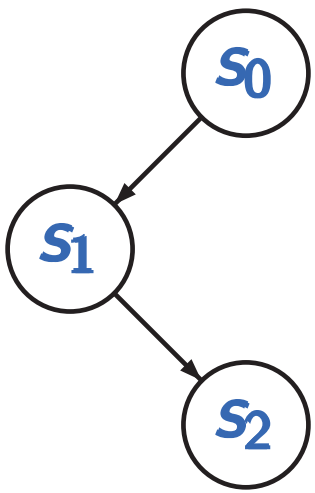
DFS forests and DFS edge-classification



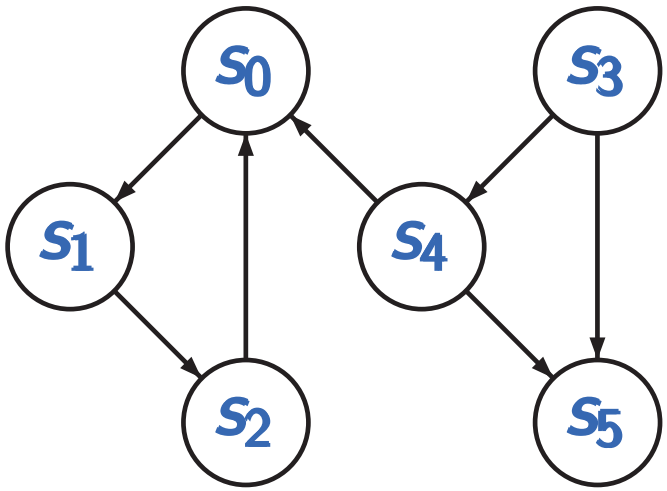
DFS forests and DFS edge-classification



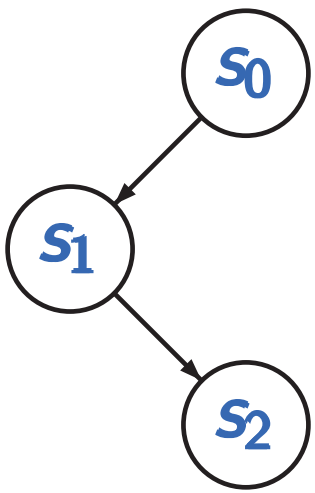
DFS-forest, e.g.,



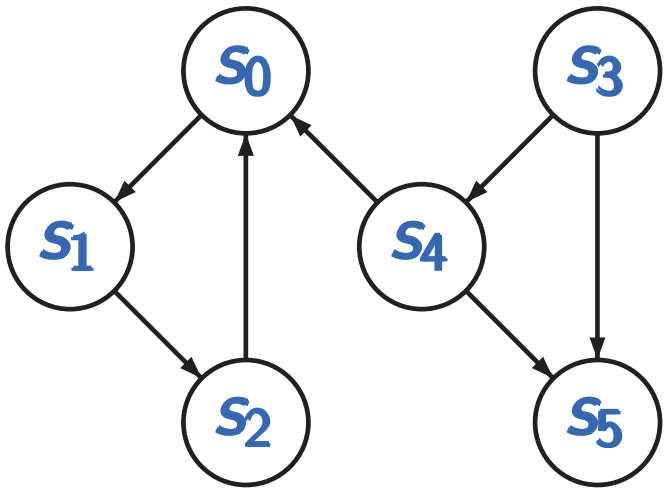
DFS forests and DFS edge-classification



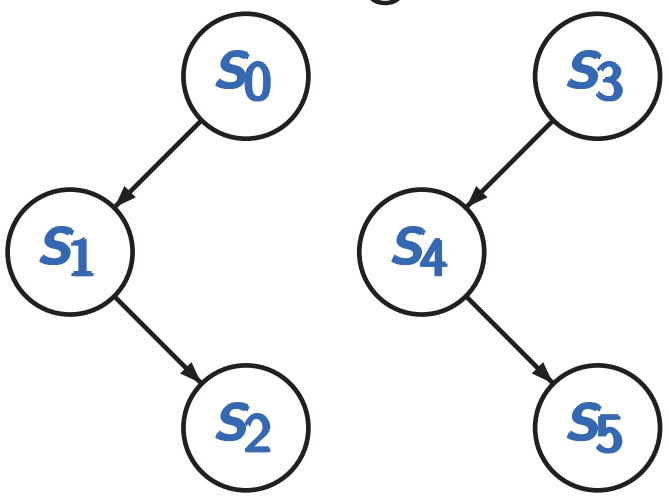
DFS-forest, e.g.,



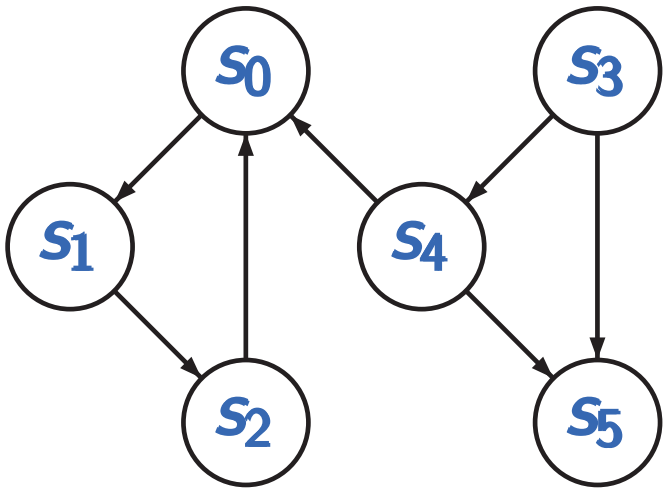
DFS forests and DFS edge-classification



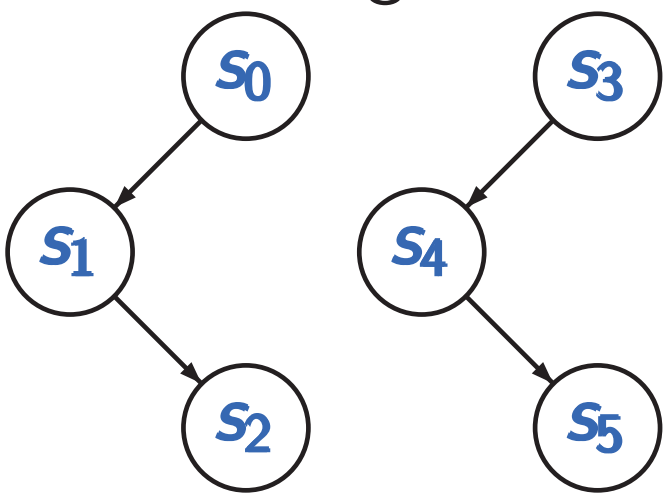
DFS-forest, e.g.,



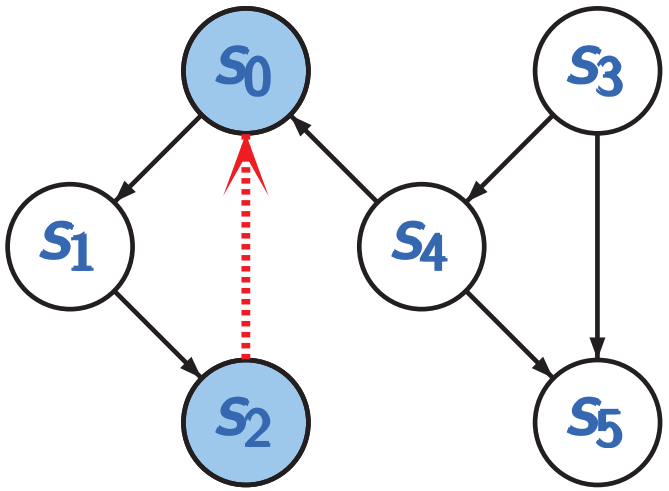
DFS forests and DFS edge-classification



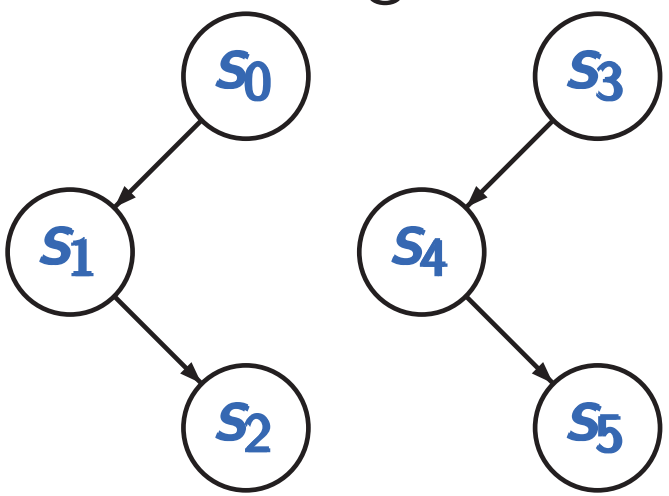
DFS-forest, e.g.,



DFS forests and DFS edge-classification



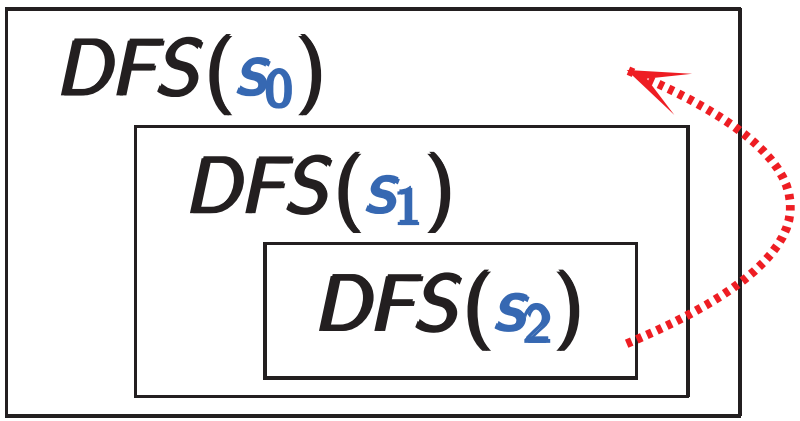
DFS-forest, e.g.,



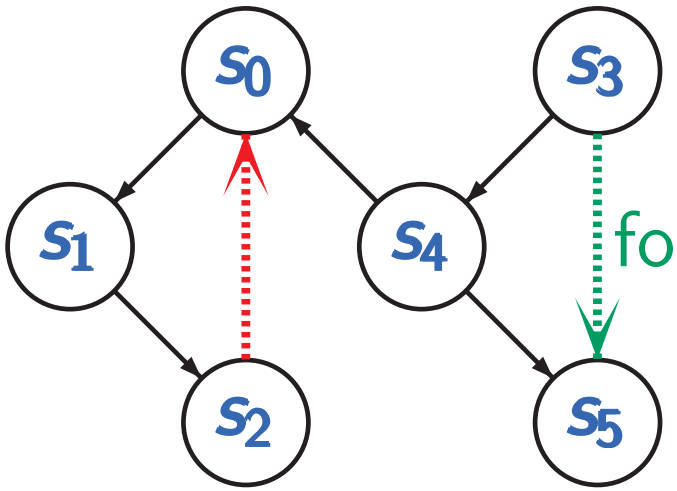
backward edge

$s_2 \longrightarrow s_0$

"closes" a cycle

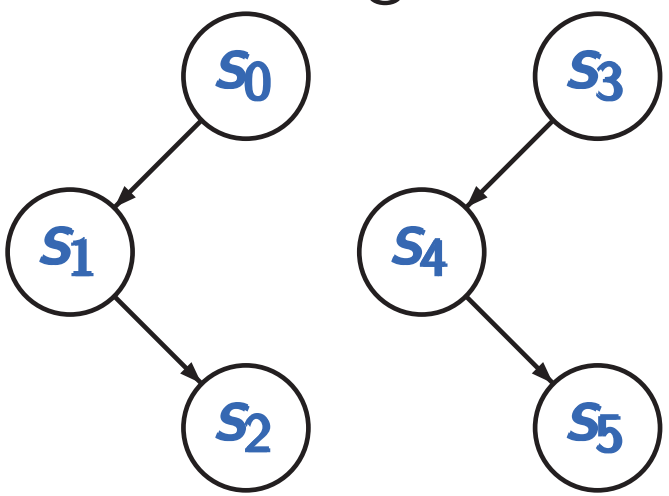


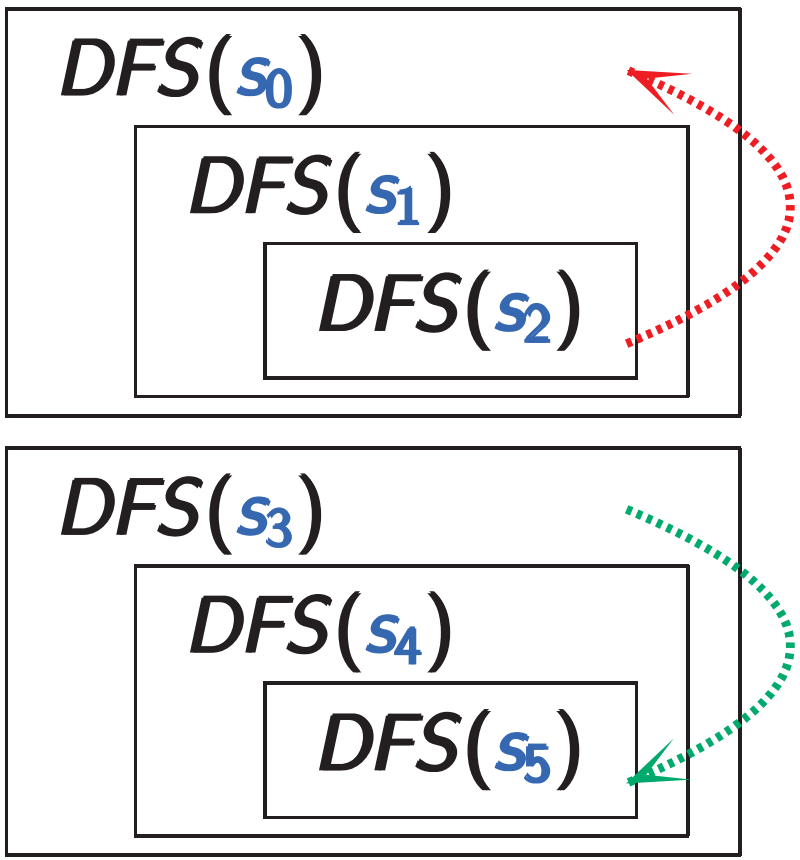
DFS forests and DFS edge-classification



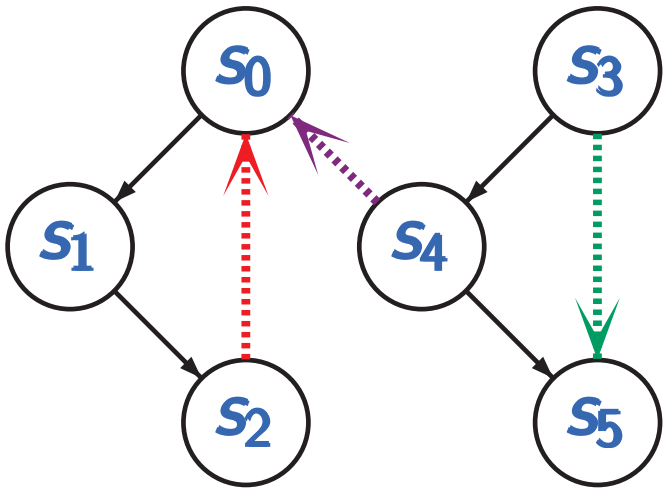
forward-edge

DFS-forest, e.g.,

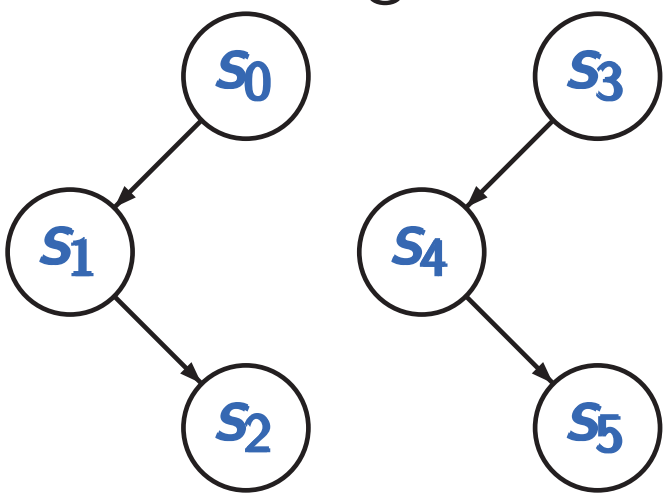




DFS forests and DFS edge-classification



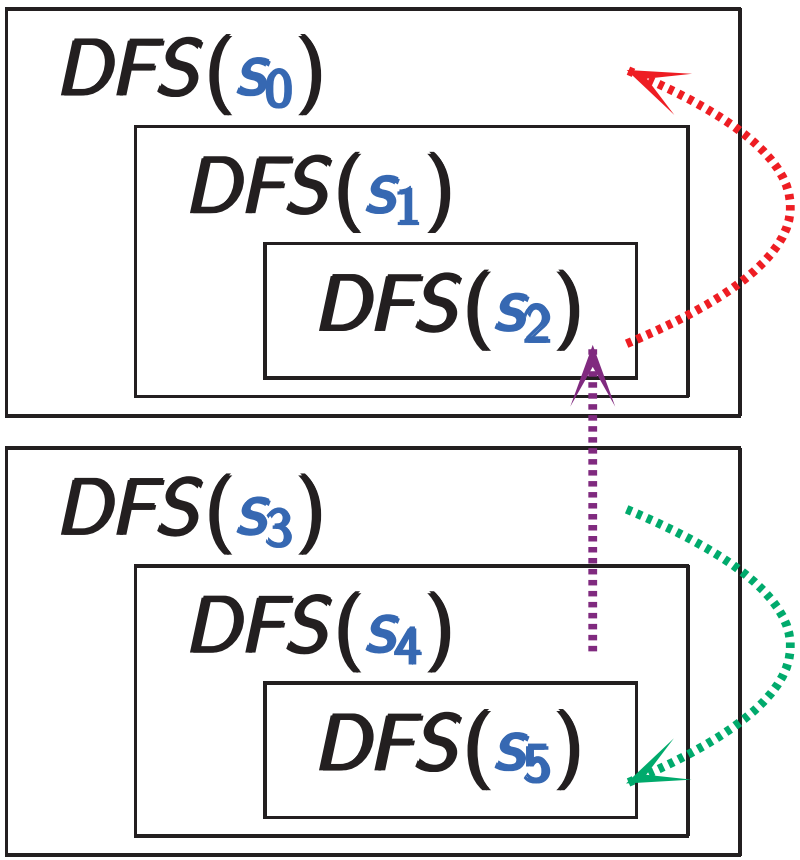
DFS-forest, e.g.,



cross edge

$s_4 \longrightarrow s_0$

connects two (sub)trees



DFS-based cycle check

DFS-based cycle check

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

The following statements are equivalent:

(1) G is cyclic

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

DFS-based cycle check

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

The following statements are equivalent:

(1) G is cyclic

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

Cycle check in digraphs:

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

DFS-based cycle check

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

The following statements are equivalent:

(1) G is cyclic

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

Cycle check in digraphs:

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

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

DFS-based cycle check

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

> The following statements are equivalent:
>
> (1) s belongs to a cycle $s\, s_1\, s_2 \ldots s_k\, s$
>
> (2) The DFS started with s finds a backward edge $s' \longrightarrow s$.

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

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

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

DFS-based persistence checking

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

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

DFS-based persistence checking

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

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

initially all states are unmarked

REPEAT

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

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

DFS-based persistence checking

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

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

initially all states are unmarked

1. DFS: visits all reachable states

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

DFS-based persistence checking

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

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

initially all states are unmarked

REPEAT

1. DFS: visits all reachable states

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

mark s;

IF *CYCLE_CHECK*(s) THEN

return "no"

FI

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

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

return "yes"

Persistence checking ⟵ Nested DFS

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

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

initially all states are unmarked

REPEAT

1. DFS: visits all reachable states

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

  mark s;

  IF *CYCLE_CHECK*(s) THEN

    return "no"

  FI

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

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

return "yes"

Time complexity of nested DFS

Time complexity of nested DFS

REPEAT

1. DFS: visits all reachable states

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

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

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

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

Time complexity of nested DFS

REPEAT

1. DFS: visits all reachable states

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

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

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

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

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

Time complexity of nested DFS

REPEAT

1. DFS: visits all reachable states

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

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

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

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

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

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

Time complexity of nested DFS

1. DFS: visits all reachable states

REPEAT

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

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

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

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

Nested DFS

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

Nested DFS

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

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

checks whether s belongs to a cycle

Nested DFS

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

1. DFS: visits all reachable states

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

checks whether s belongs to a cycle

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

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

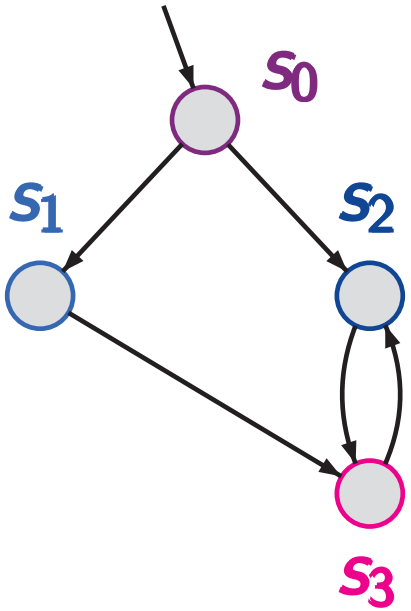
1. DFS: visits all reachable states

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**

Example: nested DFS

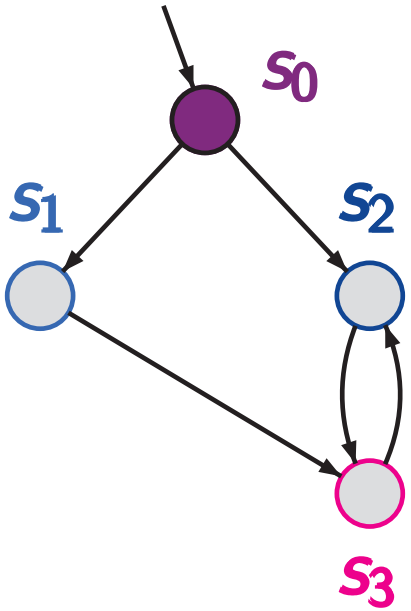


$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



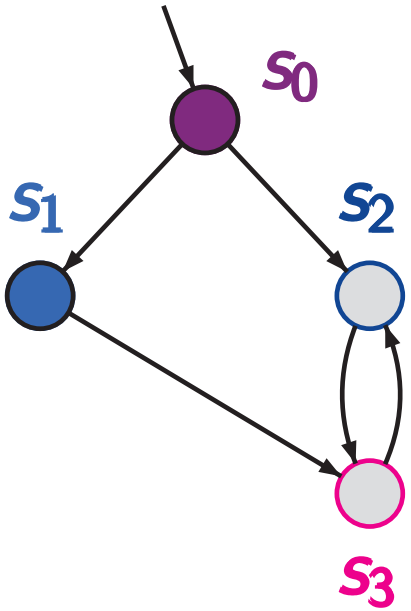
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

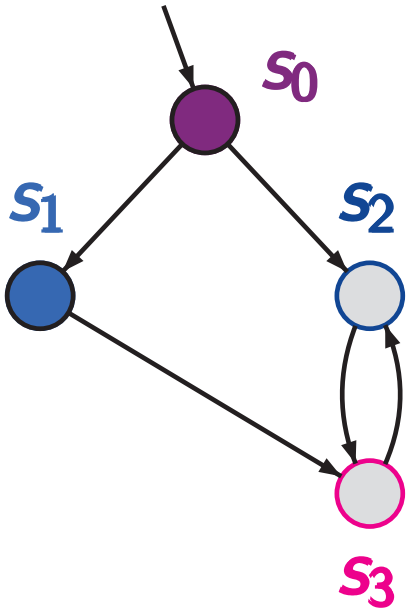
$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

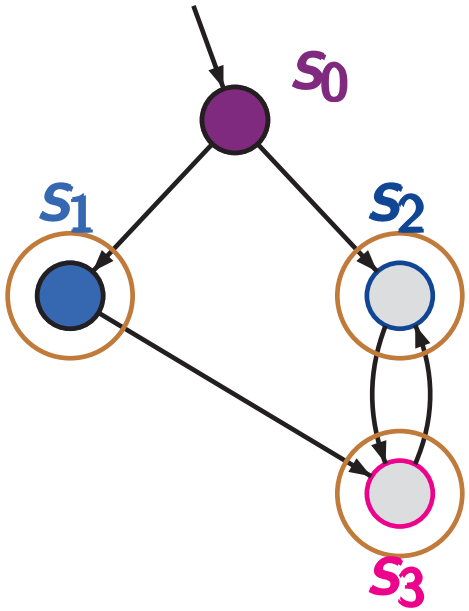
$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

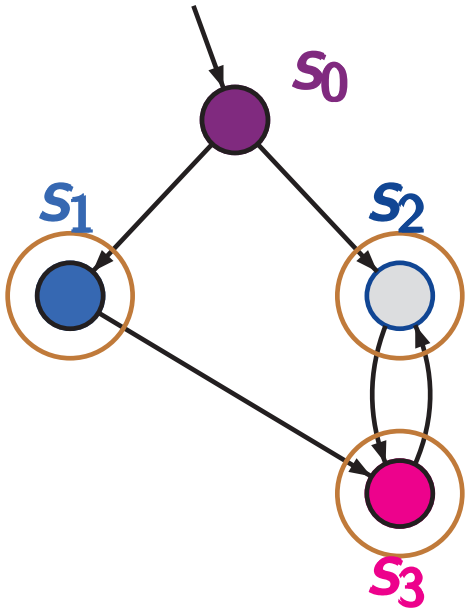
$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

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

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

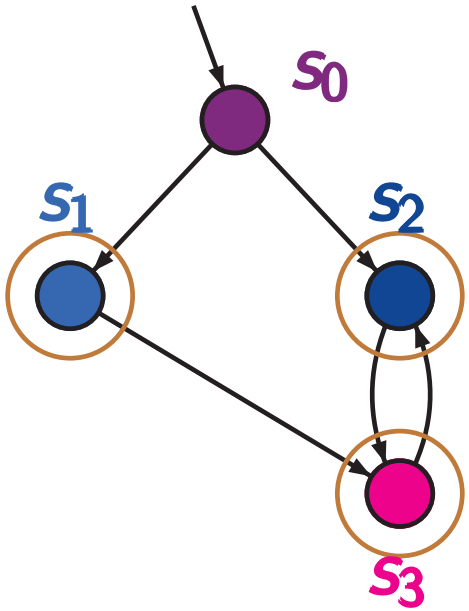
$CYCLE\_CHECK(s_1)$

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

$DFS(s_3)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

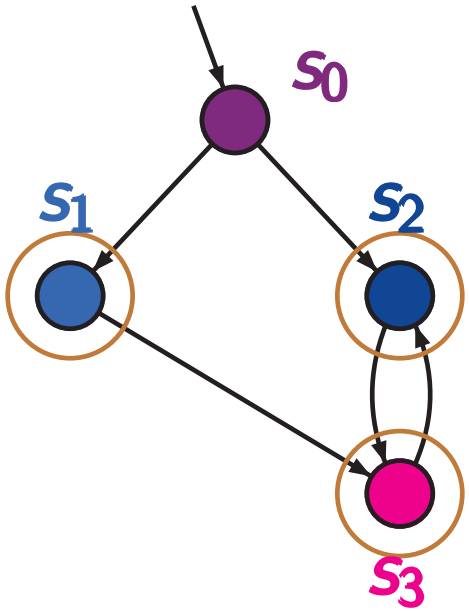
$V = \{s_1, s_2, s_3\}$

$DFS(s_3)$

$DFS(s_2)$

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

Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

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

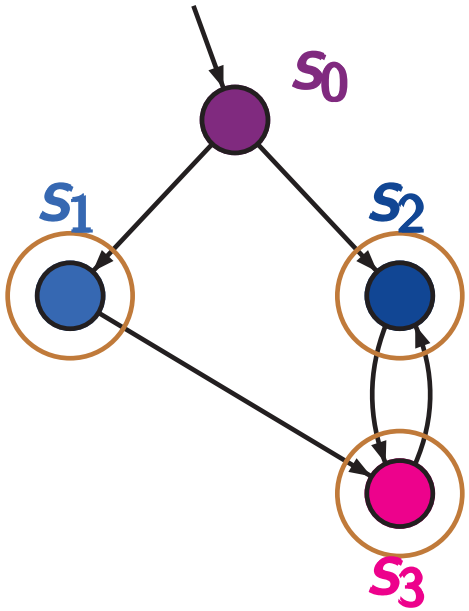
$DFS(s_3)$

$DFS(s_2)$

$CYCLE\_CHECK(s_2)$

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

Example: nested DFS ⟵ fails



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

$DFS(s_0)$

$DFS(s_1)$

$CYCLE\_CHECK(s_1)$

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

$DFS(s_3)$

$DFS(s_2)$

$CYCLE\_CHECK(s_2)$

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

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

Nested DFS

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

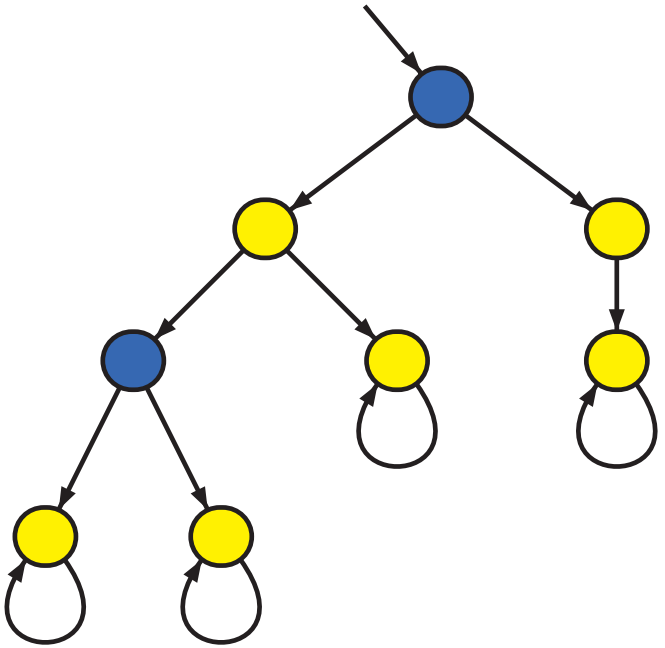
1. **DFS:** visits all reachable states
2. **DFS:** *CYCLE_CHECK*(s) for states s with $s \not\models a$
   - checks whether s belongs to a cycle by searching a backward edge $s' \rightarrow s$
   - ignores states that have been visited in previous calls of *CYCLE_CHECK* by using a global visiting set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

Nested DFS

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

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

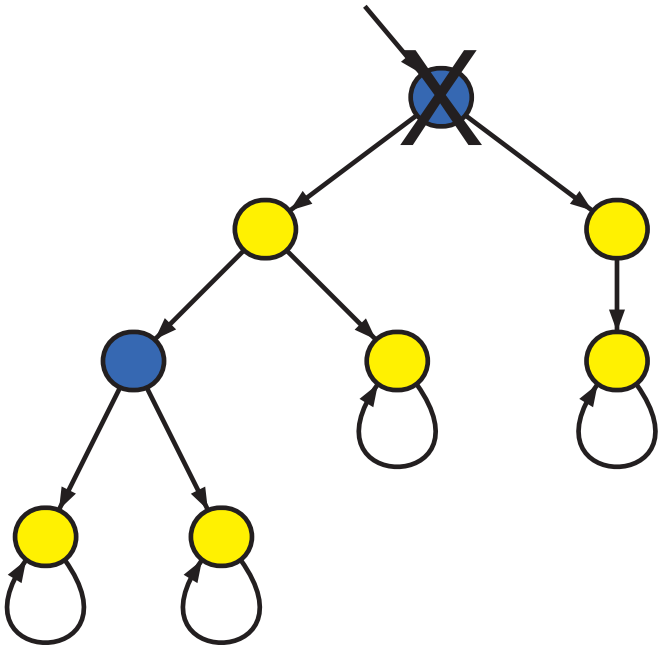
Nested DFS



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

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

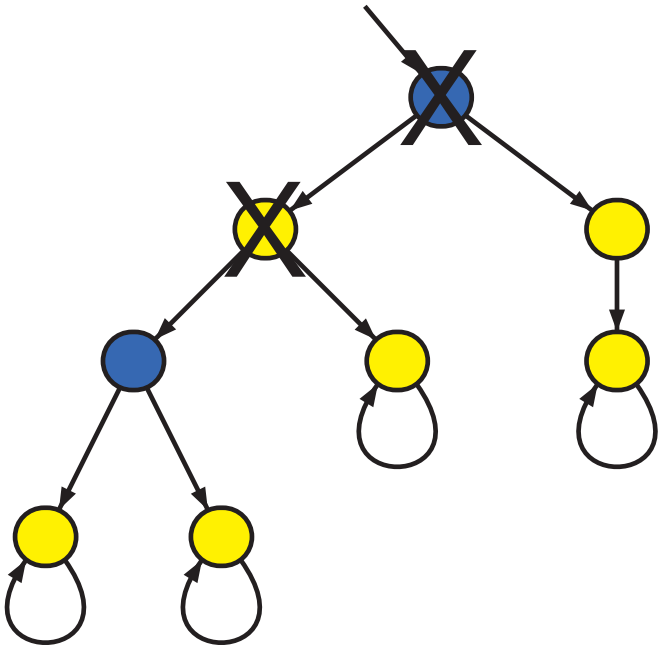
Nested DFS



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

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

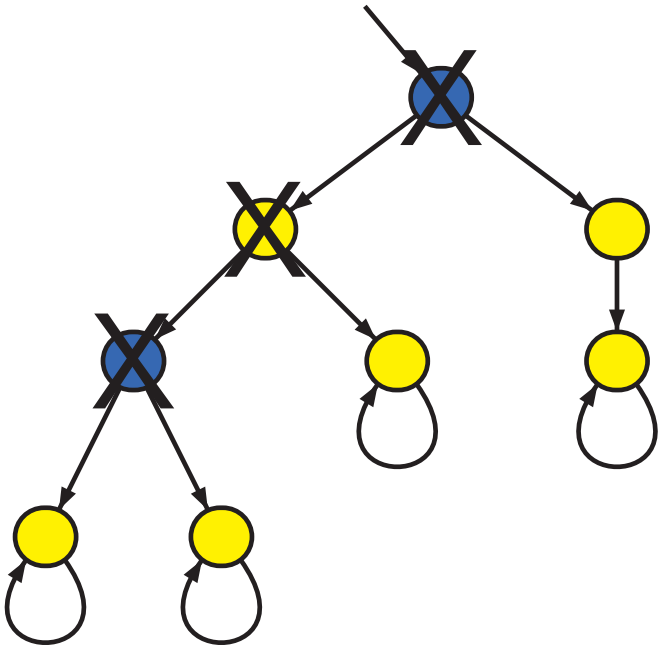
Nested DFS



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

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

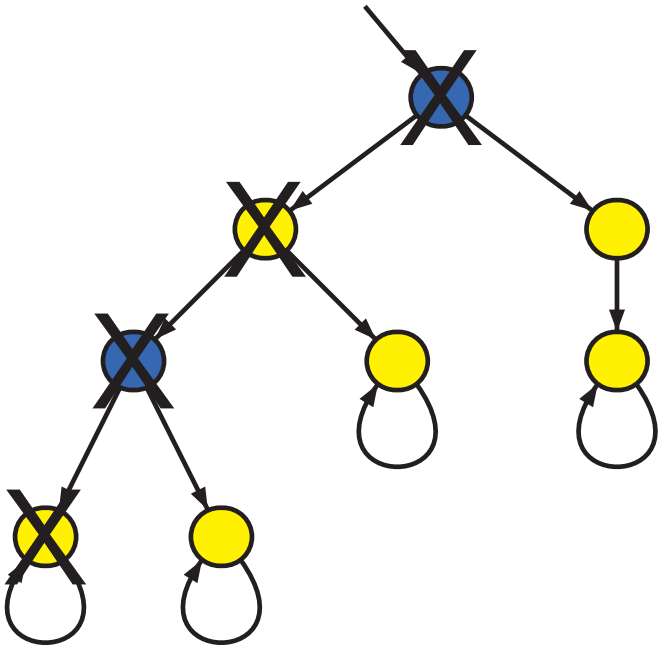
Nested DFS



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

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

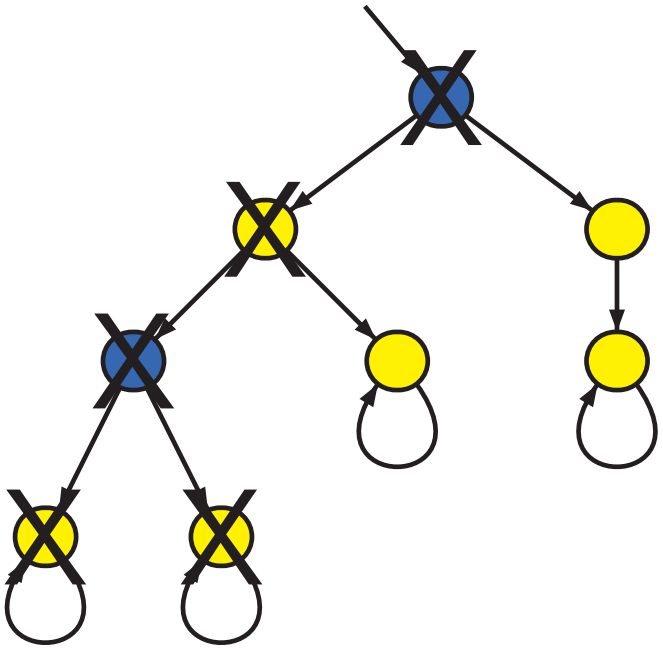
Nested DFS



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

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

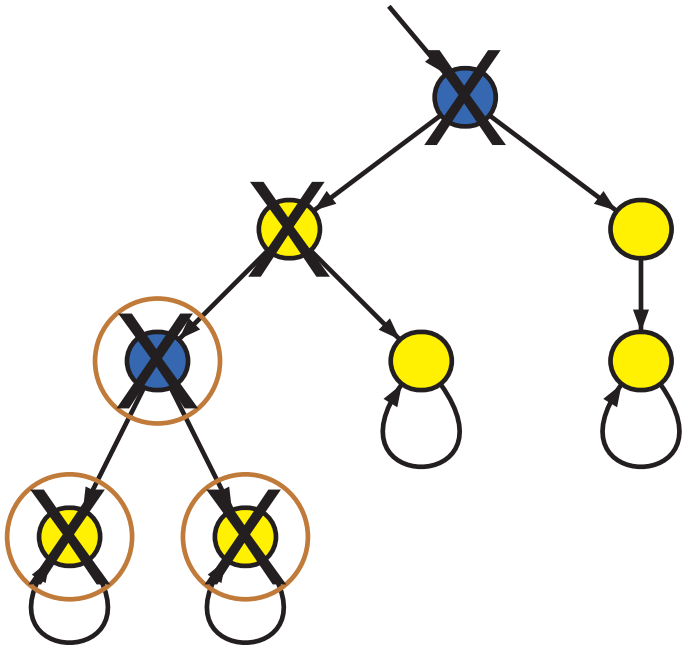
Nested DFS



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

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

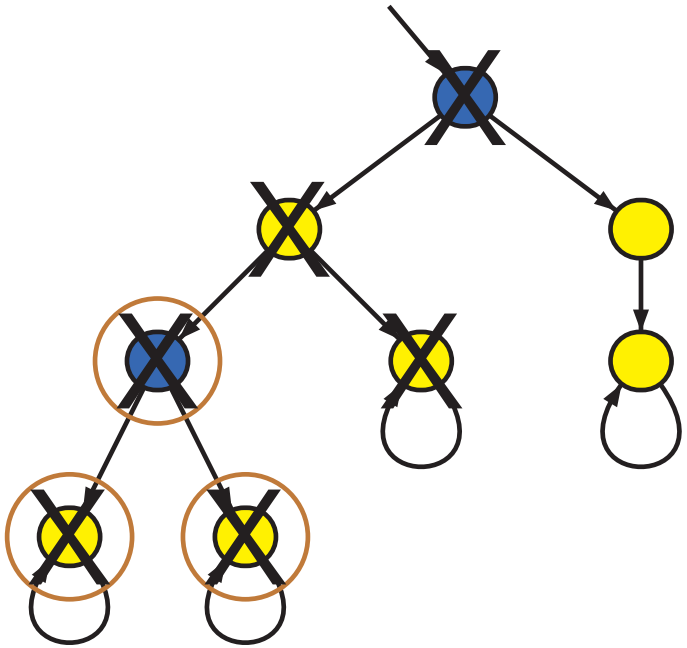
Nested DFS



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

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

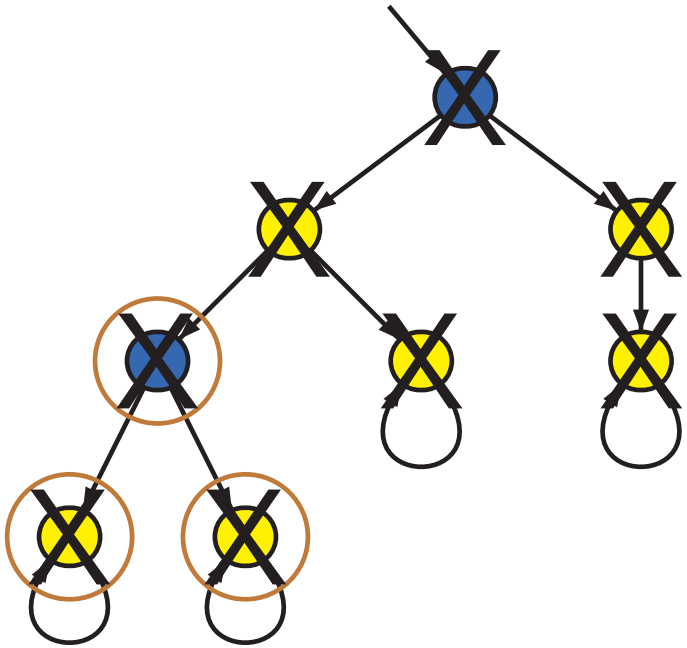
Nested DFS



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

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

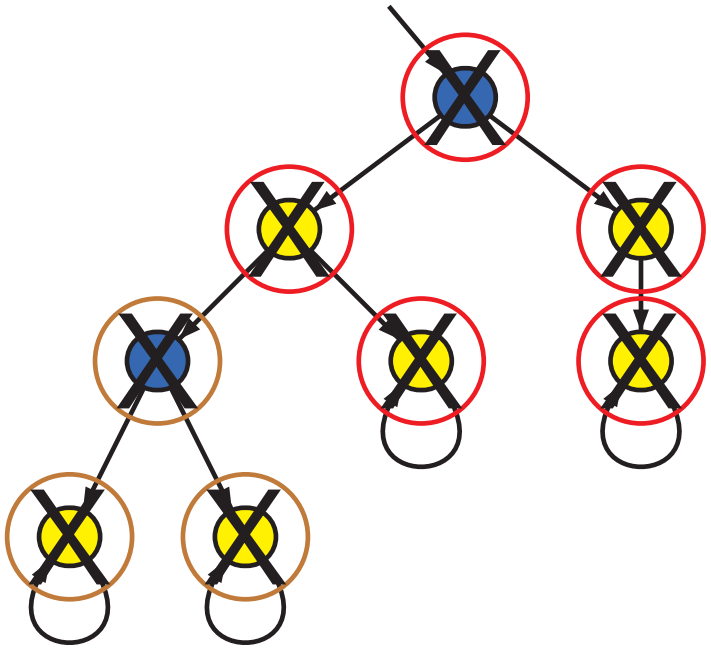
Nested DFS



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

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

Nested DFS



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

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

Nested DFS (pseudo code)

$U := \emptyset$;

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

Nested DFS (pseudo code)

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

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

Nested DFS (pseudo code)

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

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

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

pseudo code for $DFS(s)$

Nested DFS (pseudo code)

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

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

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

pseudo code for $DFS(s)$

Nested DFS (pseudo code)

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

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

pseudo code for $DFS(s)$

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

Nested DFS (pseudo code)

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

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

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

pseudo code for $DFS(s)$

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

Nested DFS (pseudo code)

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

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

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

pseudo code for $DFS(s)$

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

Nested DFS (pseudo code)

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

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

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

pseudo code for $DFS(s)$

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

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

Nested DFS (pseudo code)

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

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

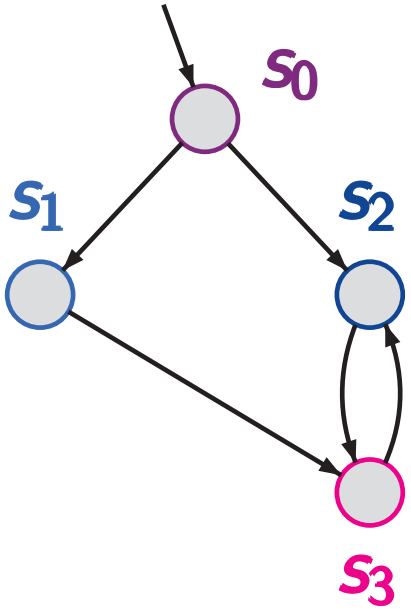
FOR ALL $s_0 \in S_0$ DO $DFS(s_0)$ OD ;

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

pseudo code for $DFS(s)$

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

Example: nested DFS

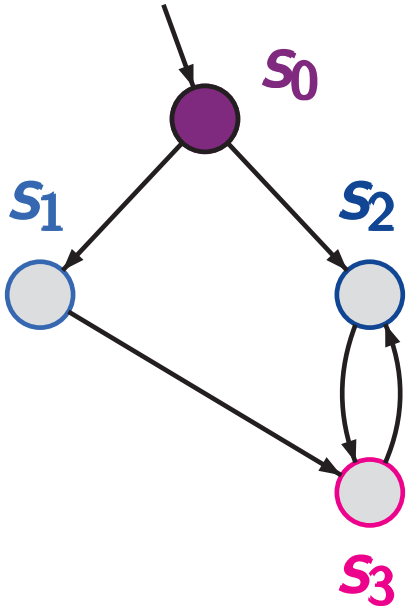


$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



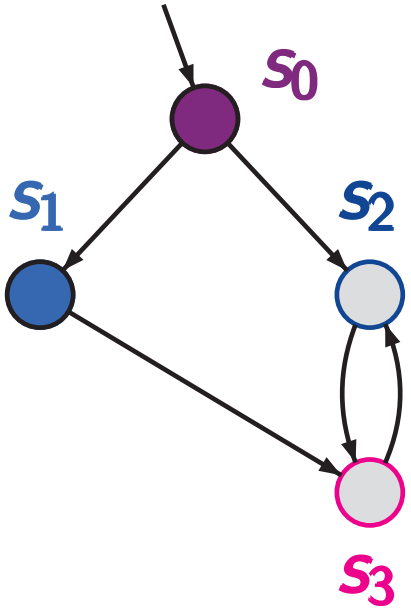
$DFS(s_0)$

$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



$DFS(s_0)$

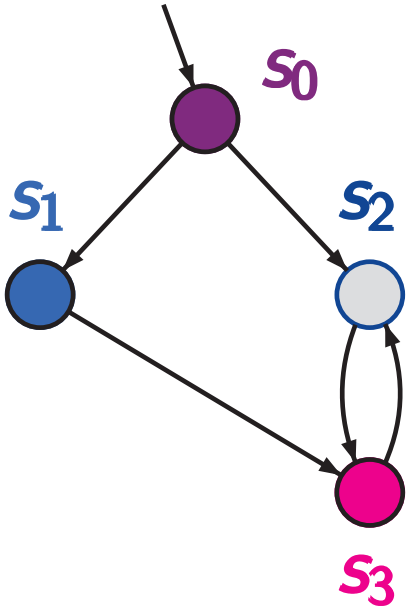
$DFS(s_1)$

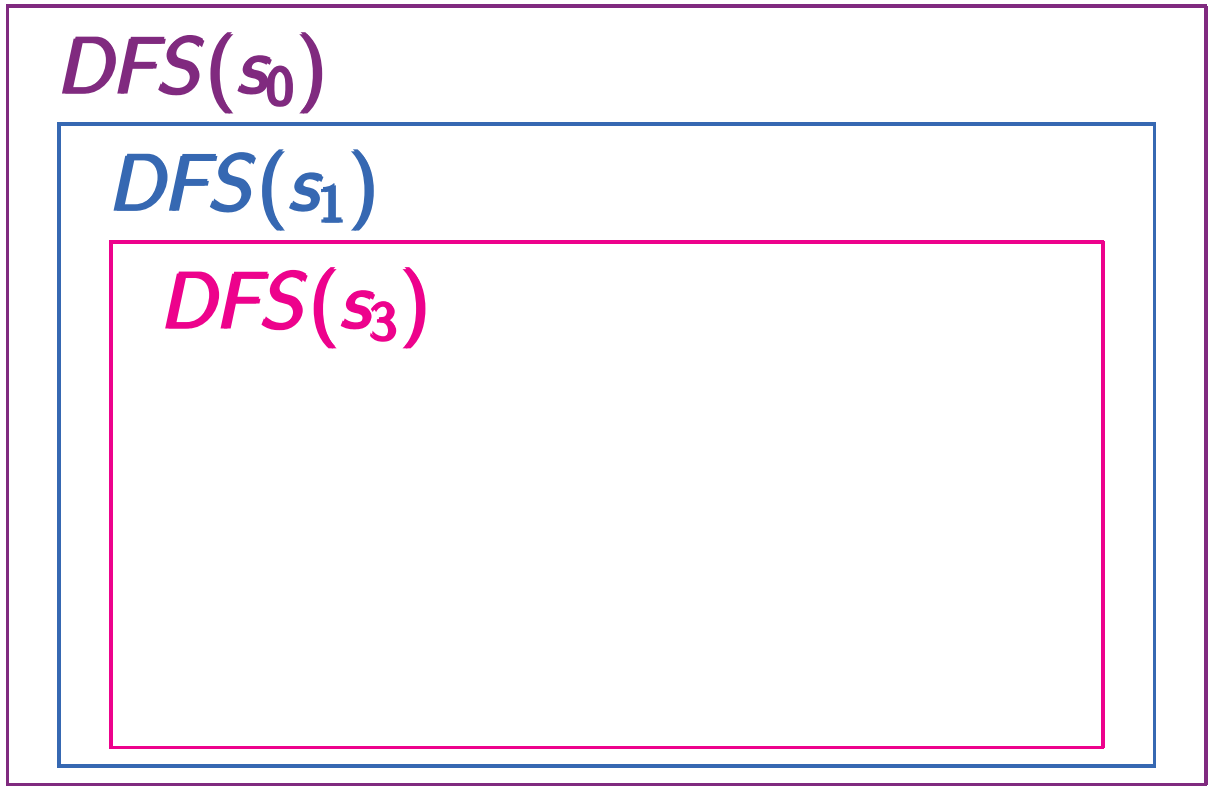
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



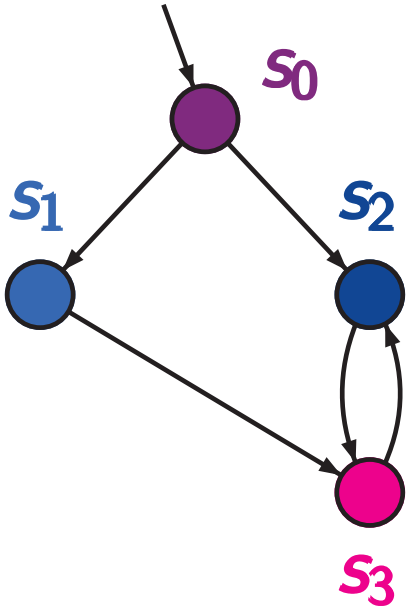


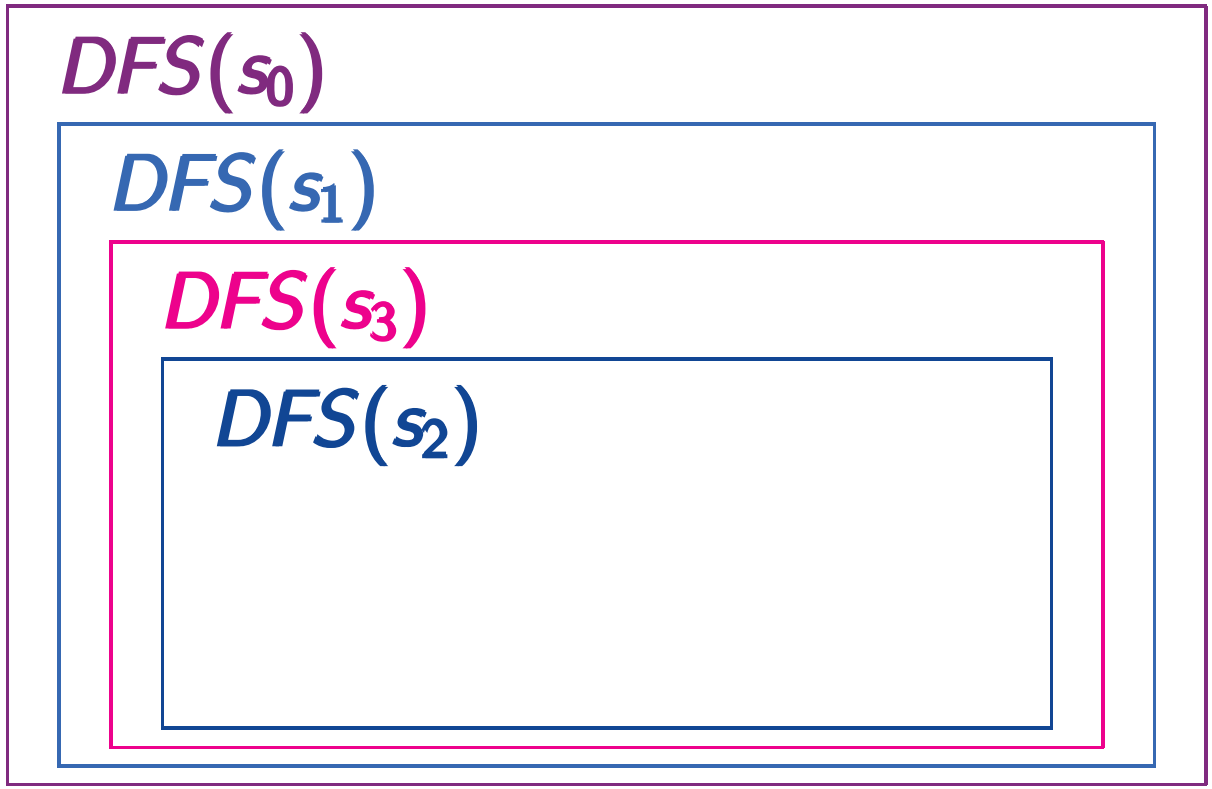
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



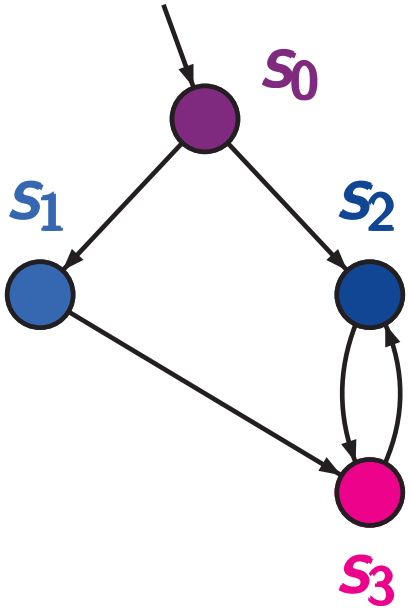


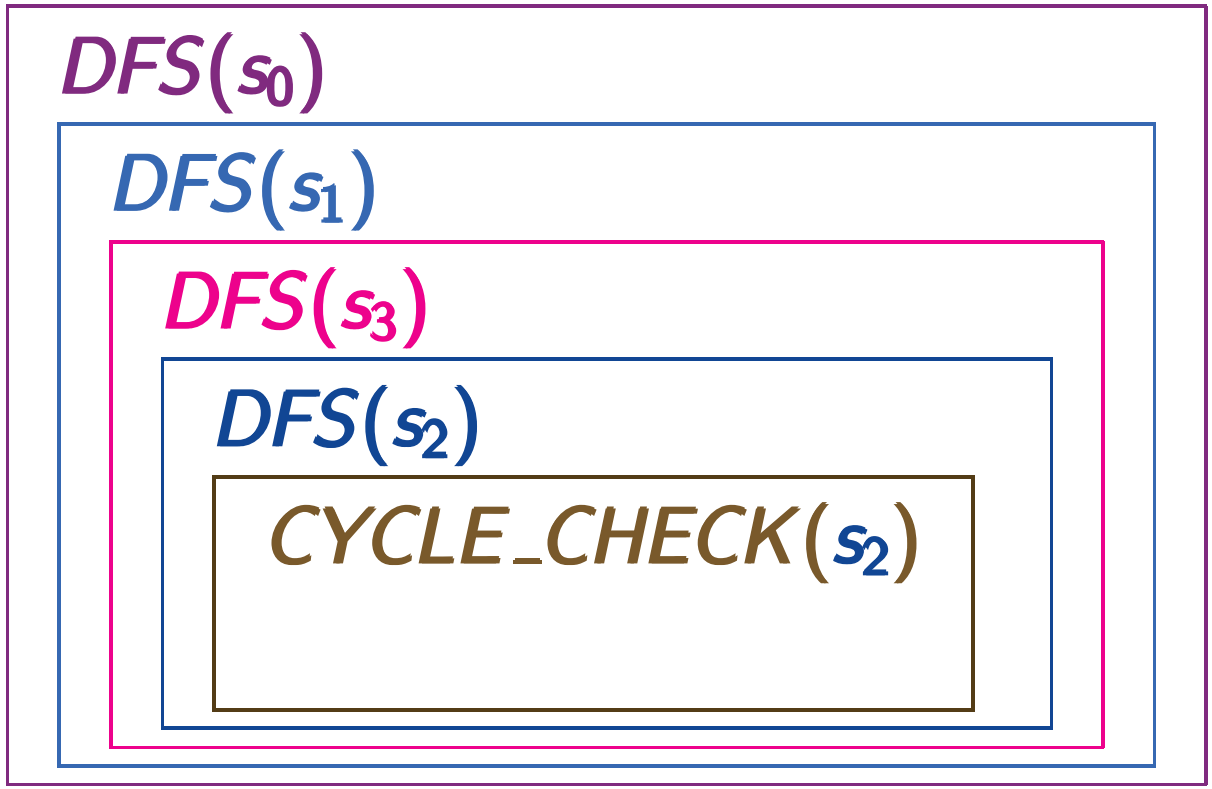
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



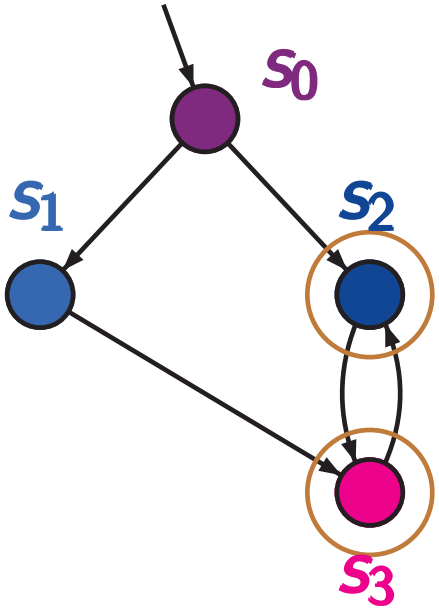


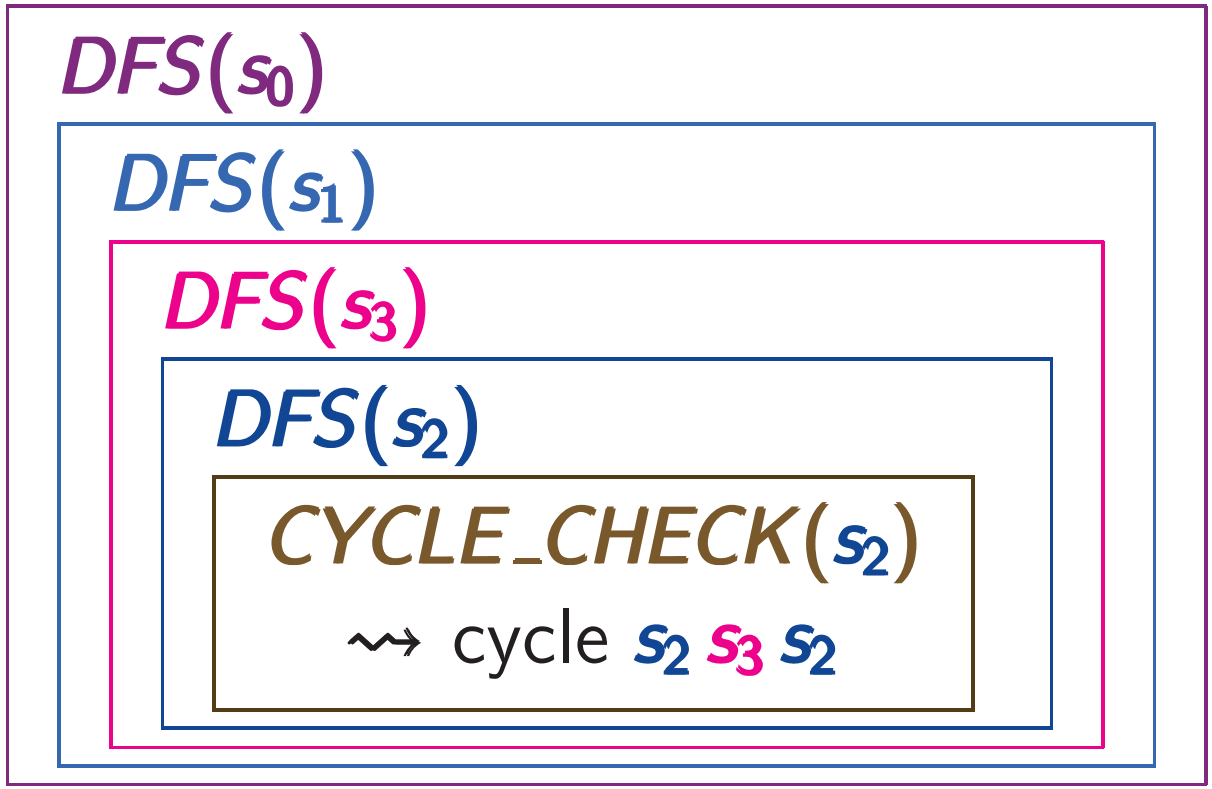
$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

Example: nested DFS



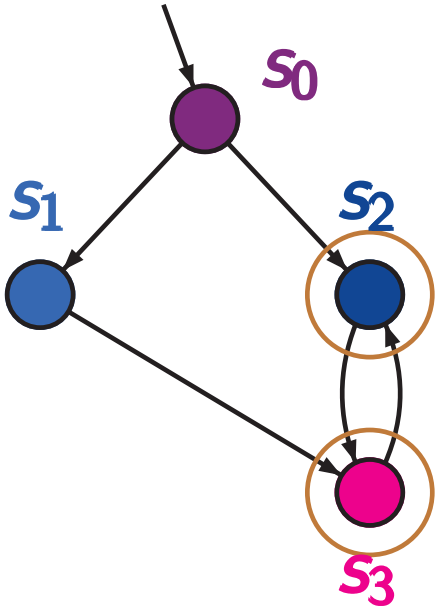


$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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

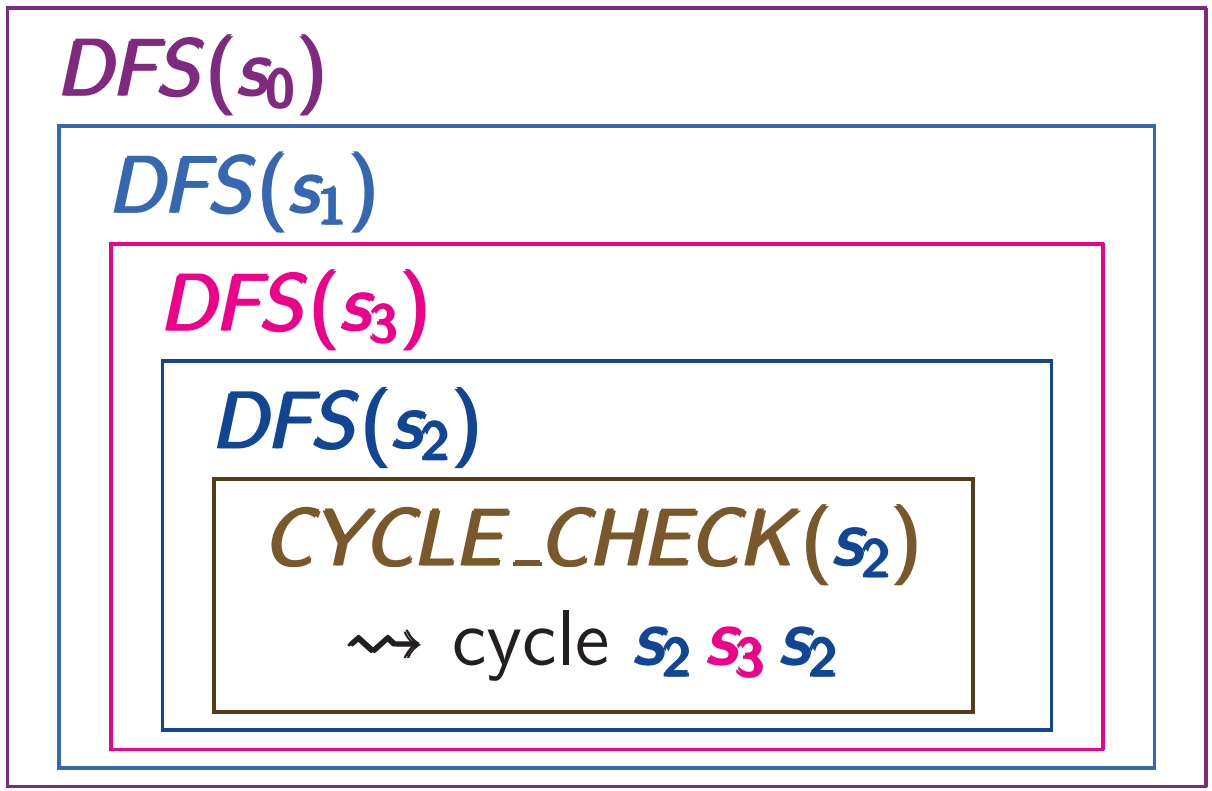
Example: nested DFS



$s_1, s_2 \not\models a$

$s_0, s_3 \models a$

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



↑
returns correct
answer "**no**"

Nested DFS with counterexample generation

Nested DFS with counterexample generation

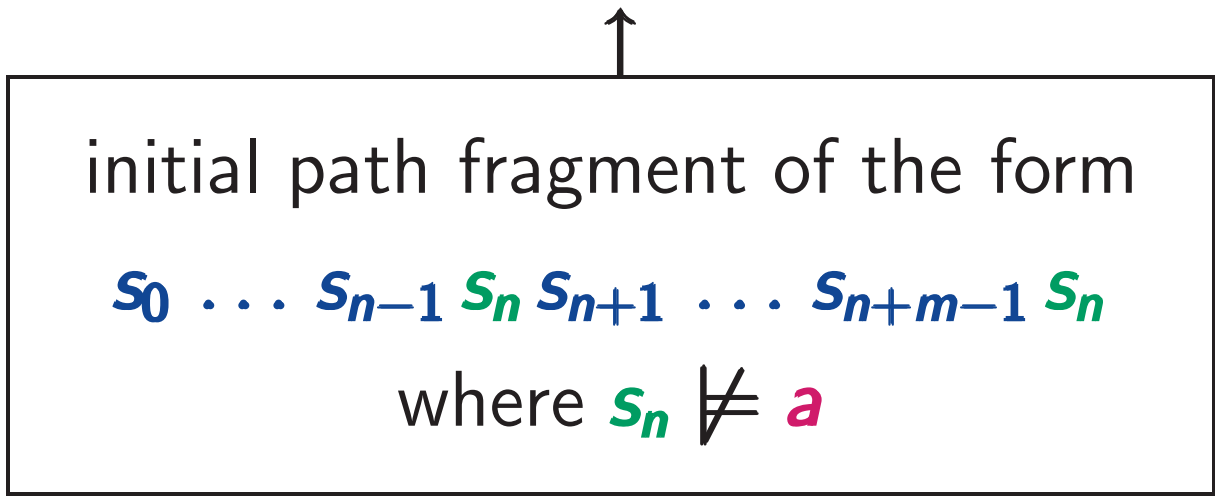
input: finite TS $\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$
persistence condition $a \in AP$

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

Nested DFS with counterexample generation

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

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



Nested DFS with counterexample generation

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

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

initial path fragment of the form

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

where $s_n \not\models a$

... iterative formulation with 2 stacks ...

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

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

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

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

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

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

WHILE $S_0 \not\subseteq U$ DO

OD

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

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

WHILE $S_0 \not\subseteq U$ DO

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

OD

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

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

WHILE $S_0 \not\subseteq U$ DO

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

OD

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

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

WHILE $S_0 \not\subseteq U$ DO

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

WHILE $\pi \neq \emptyset$ DO

$s := Top(\pi)$;

OD OD

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

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

```
WHILE 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
```

```
OD OD   FI
```

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

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

```
WHILE S0 ⊈ U DO
    choose s0 ∈ S0 \ U; insert s0 in U; Push(π, s0);
    WHILE π ≠ ∅ DO
        s := Top(π);
        IF Post(s) ⊈ U
         THEN choose s' ∈ Post(s) \ U;
              insert s' in U; Push(π, s')


    FI
OD OD
```

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

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

```
WHILE S0 ⊈ U DO
    choose s0 ∈ S0 \ U; insert s0 in U; Push(π, s0);
    WHILE π ≠ ∅ DO
        s := Top(π);
        IF Post(s) ⊈ U
         THEN choose s' ∈ Post(s) \ U;
              insert s' in U; Push(π, s')
         ELSE Pop(π);

    FI
OD OD
```

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

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

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

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

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

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

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

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

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

Algorithm *CYCLE_CHECK*(*s*)

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

Algorithm *CYCLE_CHECK*(s)

Push(ξ, s); insert s in V;

Algorithm *CYCLE_CHECK*(*s*)

Push(ξ, s); insert s in V;

`WHILE` $\xi \neq \varnothing$ `DO`

Algorithm *CYCLE_CHECK*(s)

Push(ξ, s); insert s in V;

WHILE $\xi \neq \varnothing$ DO

    $s' :=$ *Top*(ξ);

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

Algorithm *CYCLE_CHECK*(s)

Push(ξ, s); insert s in V;

WHILE $\xi \neq \varnothing$ DO

  $s' :=$ ***Top***(ξ);

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

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

Algorithm *CYCLE_CHECK*(s)

$Push(\xi, s)$; 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 $Post(s') \not\subseteq V$

Algorithm *CYCLE_CHECK*(s)

Push(ξ, s); insert s in V;

WHILE $\xi \neq \varnothing$ DO

  $s' :=$ *Top*(ξ);

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

    THEN *Push*(ξ, s); return "true"

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

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

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

Algorithm *CYCLE_CHECK*(s)

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

WHILE $\xi \neq \varnothing$ DO

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

  IF $s \in Post(s')$

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

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

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

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

      ELSE $Pop(\xi)$

    FI

Algorithm *CYCLE_CHECK*(s)

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

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
    FI
OD
return "false"
```

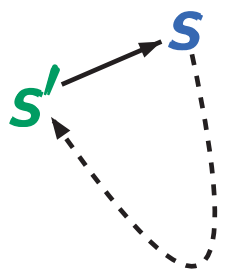
Algorithm *CYCLE_CHECK*(s)

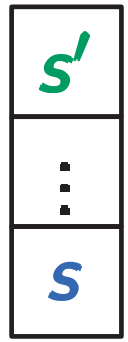
Push(ξ, s); insert s in V;

WHILE $\xi \neq \varnothing$ DO

$s' :=$ ***Top***(ξ);

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





stack ξ

THEN ***Push***(ξ, s); return "true"

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

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

insert s'' in V; ***Push***(ξ, s'');

ELSE ***Pop***(ξ)

FI

FI

OD

return "false"

Algorithm *CYCLE_CHECK*(s)

Push(ξ, s); insert s in V;

WHILE $\xi \neq \varnothing$ DO

$s' :=$ ***Top***(ξ);

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

THEN ***Push***(ξ, s) return "true"

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

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

insert s'' in V; ***Push***(ξ, s'');

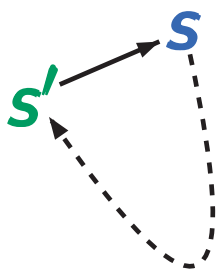
ELSE ***Pop***(ξ)

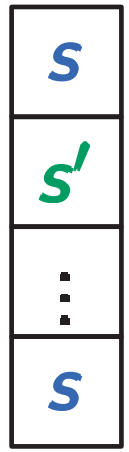
FI

FI

OD

return "false"





stack ξ

Nested DFS with counterexample generation

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

Nested DFS with counterexample generation

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

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

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm *CYCLE_CHECK*(s)

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

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm *CYCLE_CHECK*(s)

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

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

ξ: stack for counterexample

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm *CYCLE_CHECK*(s)

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

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

ξ: stack for counterexample

soundness:
1. termination
2. partial correctness

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm *CYCLE_CHECK*(s)

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

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

ξ: stack for counterexample

soundness: 1. termination ✓

2. partial correctness

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm *CYCLE_CHECK*(s)

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

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

ξ: stack for counterexample

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

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

Soundness of the nested DFS

outer DFS: visits all reachable states s

inner DFS: algorithm $\mathit{CYCLE\_CHECK}(s)$

- 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

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

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

Soundness of the nested DFS

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

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

Soundness of the nested DFS

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

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

Proof of "$\Longleftarrow$":

Soundness of the nested DFS

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

Proof of "$\Longleftarrow$":

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

Soundness of the nested DFS

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

Proof of "$\Longleftarrow$":

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

Hence: s belongs to a cycle

Soundness of the nested DFS

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

Proof of "$\Longleftarrow$":

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

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

Soundness of the nested DFS

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

Proof of "$\Longleftarrow$":

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

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

Soundness of the nested DFS

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

Proof of "$\Longleftarrow$":

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

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

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

Soundness of the nested DFS

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

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

Proof of "$\Longrightarrow$":

Soundness of the nested DFS

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

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

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

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

global visiting set of the inner DFS

Soundness of the nested DFS

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

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

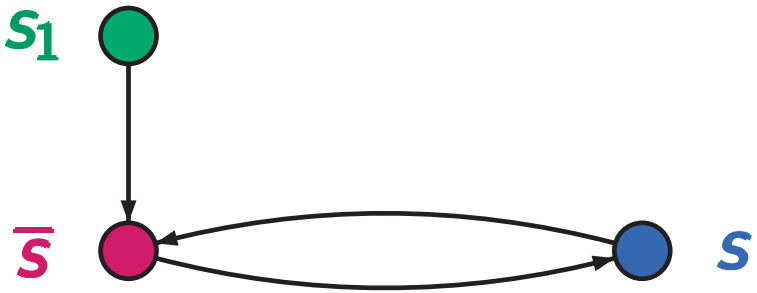
When *CYCLE_CHECK*(s) is called then there is no cycle $t_0\, t_1 \ldots t_k$ in $\mathcal{T}$ s.t. $s = t_0 = t_k$ and

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

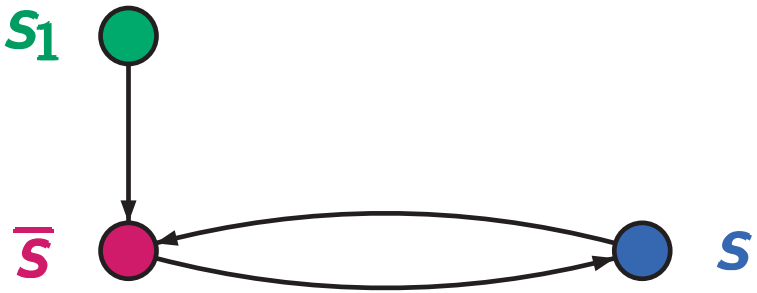
global visiting set of the inner DFS

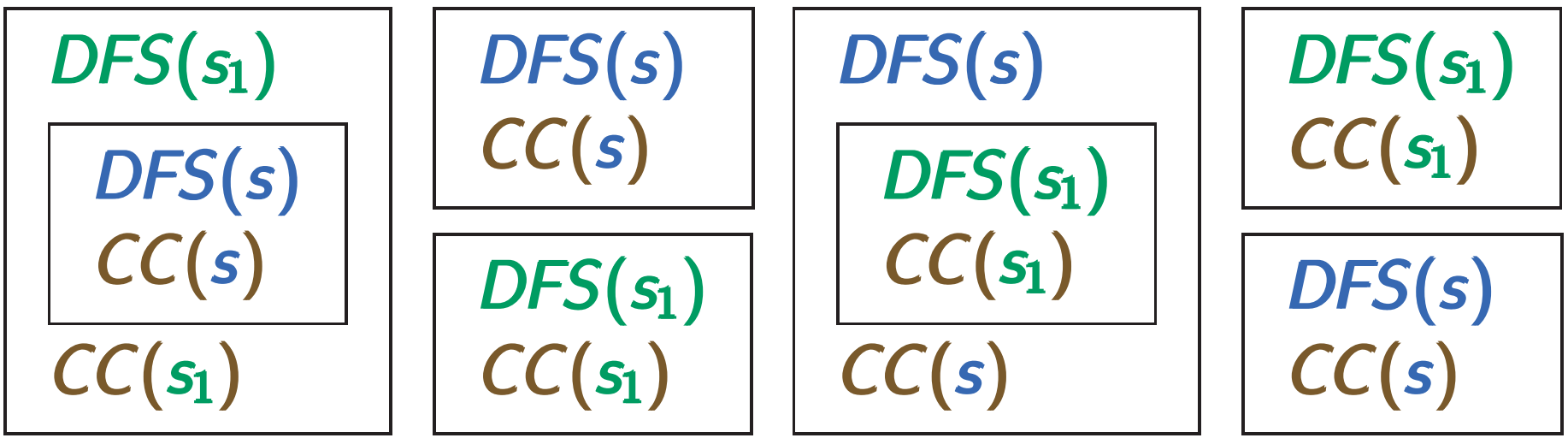
Hence: if s belongs to a cycle then *CYCLE_CHECK*(s) will find a backward edge $t \longrightarrow s$

Correctness of the nested DFS

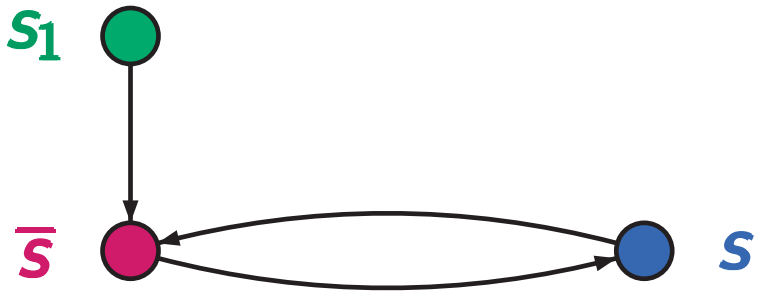


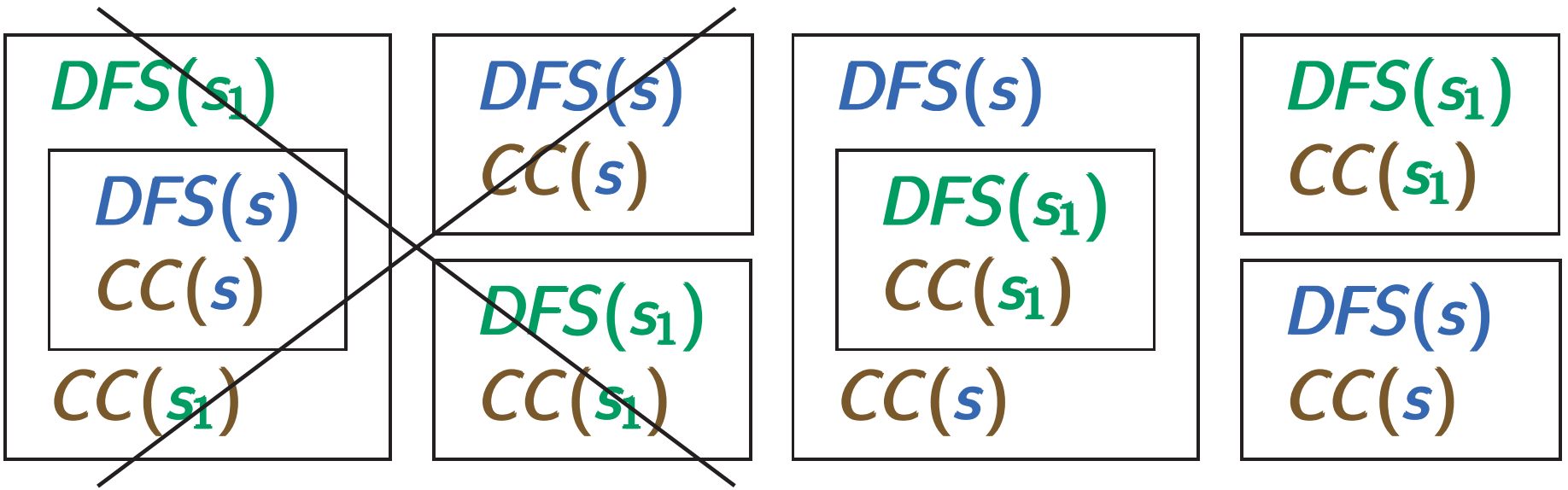
Correctness of the nested DFS





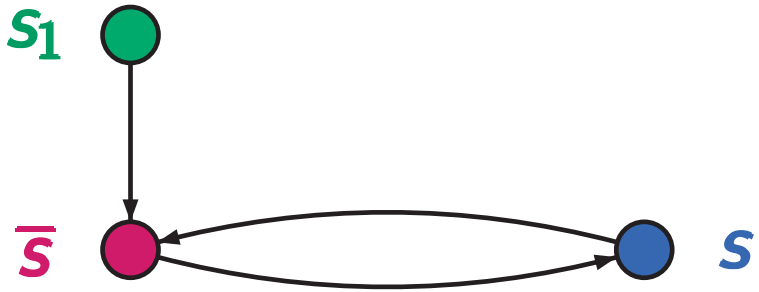
Correctness of the nested DFS

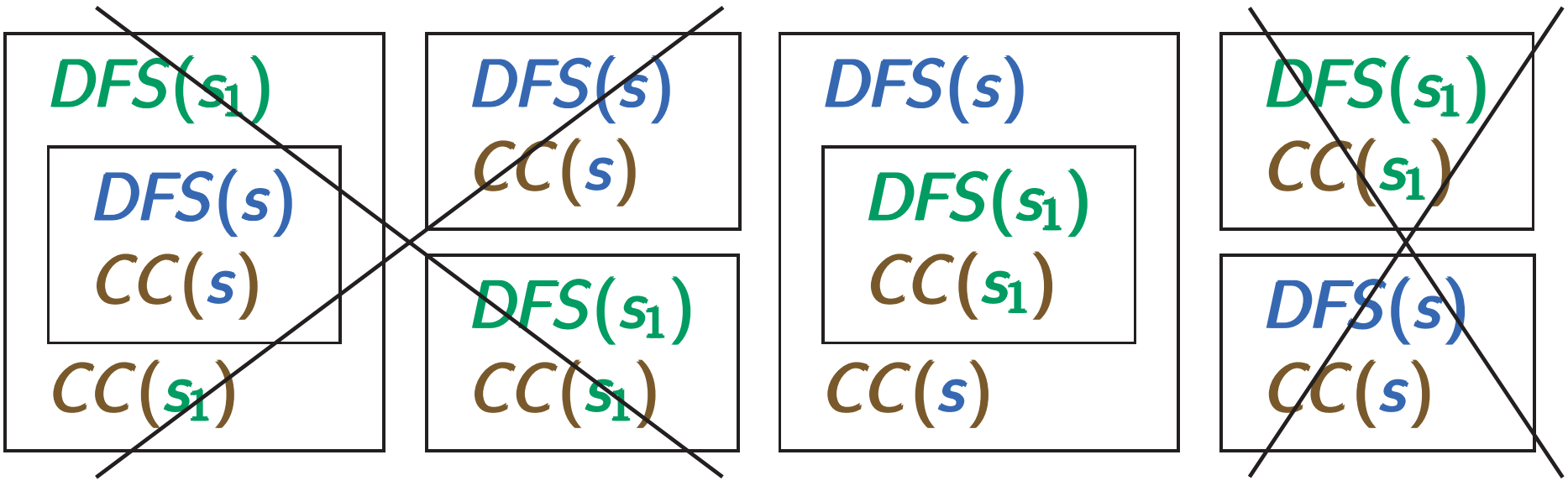




as $CC(s_1)$ is called
before $CC(s)$

Correctness of the nested DFS

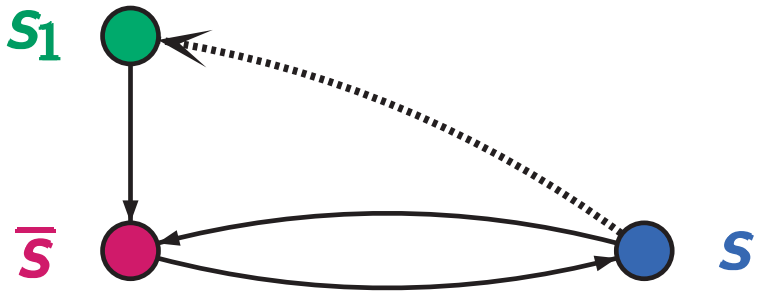




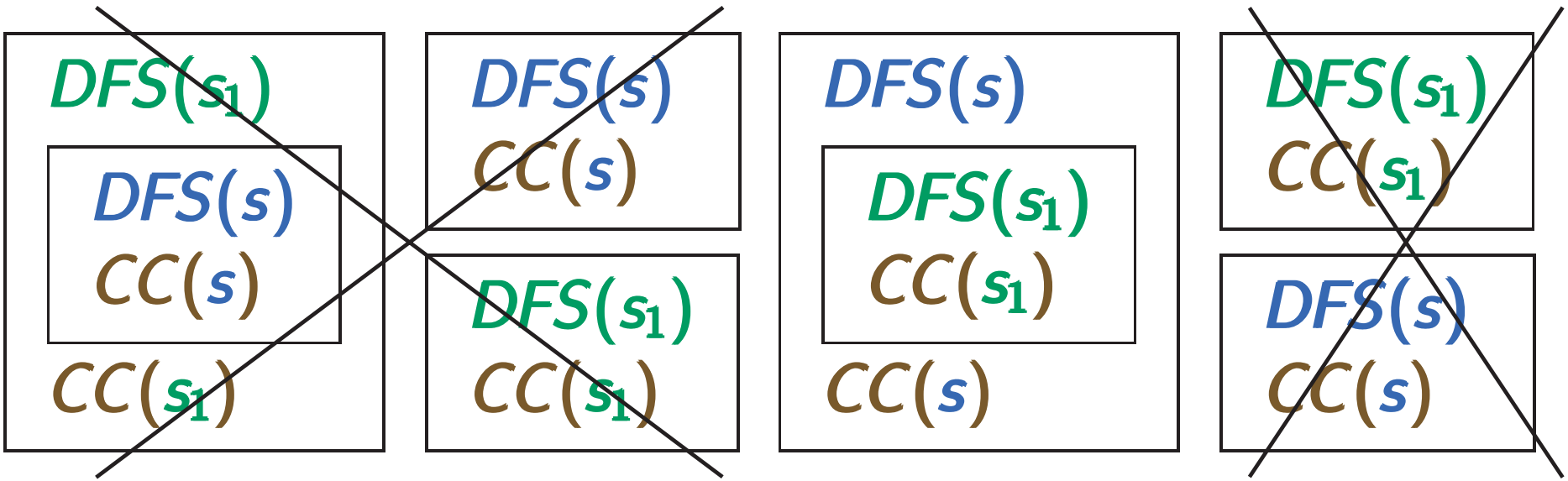
as $CC(s_1)$ is called before $CC(s)$

as s is reachable from s_1

Correctness of the nested DFS



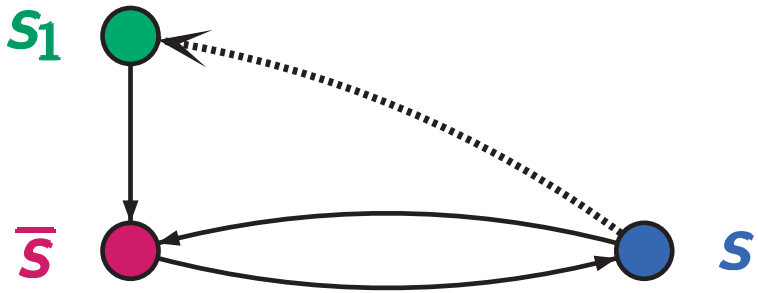
s_1 reachable from s



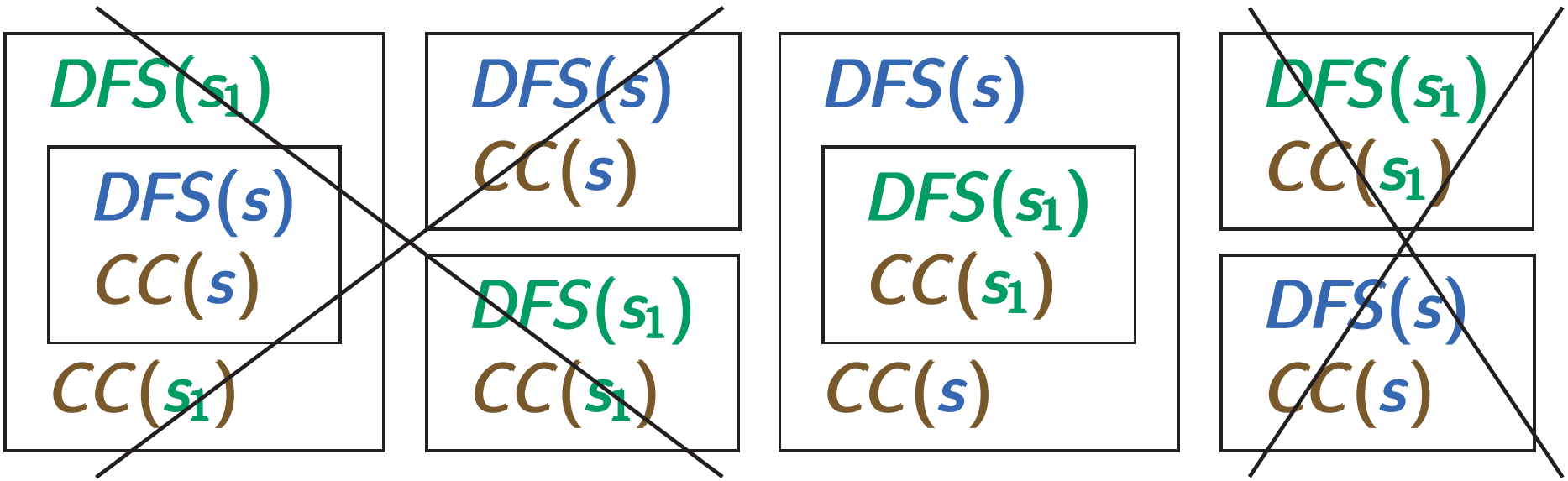
as $CC(s_1)$ is called before $CC(s)$

as s is reachable from s_1

Correctness of the nested DFS



s_1 reachable from s



as $CC(s_1)$ is called before $CC(s)$

$CC(s_1)$ would have found the cycle

$s_1 \rightsquigarrow \overline{s} \rightsquigarrow s \rightsquigarrow s_1$

as s is reachable from s_1

Further improvements of the nested DFS

Further improvements of the nested DFS

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

Further improvements of the nested DFS

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

on the fly construction

Further improvements of the nested DFS

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

on the fly construction
hash techniques for U and V

Further improvements of the nested DFS

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

on the fly construction
hash techniques for U and V

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

Further improvements of the nested DFS

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

on the fly construction

hash techniques for U and V

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

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