Verifying ω-Regular Properties

Lecture #11 of Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

May 9, 2007

Overview Lecture #11

⇒ Checking Regular Safety Properties

- Checking ω-Regular Properties
  - persistence properties
  - reduction to checking persistence properties
  - checking persistence properties
- Nested depth-first search
- Summary of regular properties

Regular safety properties

Safety property P_{safe} over *AP* is *regular*

if its set of bad prefixes is a regular language over 2^{AP}

Basic idea of the algorithm

$$TS \not\models P_{safe} \quad \text{if and only if} \quad Traces_{fin}(TS) \cap \underbrace{BadPref(P_{safe})}_{\overline{P_{safe}}} \neq \varnothing$$

$$\text{if and only if} \quad Traces_{fin}(TS) \cap \mathcal{L}(\mathcal{A}) \neq \varnothing$$

$$\text{if and only if} \quad TS \otimes \mathcal{A} \not\models \underbrace{\text{“always”} \neg F}_{\text{invariant property}}$$

$\Rightarrow$ *checking regular safety properties is reduced to invariant checking!*

Verifying regular safety properties

Let *TS* over *AP* and NFA $\mathcal{A}$ with alphabet 2^{AP} as before, regular safety property P_{safe} over *AP* such that $\mathcal{L}(\mathcal{A})$ is the set of bad prefixes of P_{safe}

The following statements are equivalent:

(a) $TS \models P_{safe}$

(b) $Traces_{fin}(TS) \cap \mathcal{L}(\mathcal{A}) = \varnothing$

(c) $TS \otimes \mathcal{A} \models P_{inv(A)}$

where $P_{inv(A)}$ = *"always"* $\neg F$

Overview Lecture #11

- Checking Regular Safety Properties

$\Rightarrow$ Checking ω-Regular Properties
  - persistence properties
  - reduction to checking persistence properties
  - checking persistence properties

- Nested depth-first search

- Summary of regular properties

ω-regular properties

LT property P over *AP* is *ω-regular*

if P is an ω-regular language over 2^{AP}

Basic idea of the algorithm

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

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

Persistence property

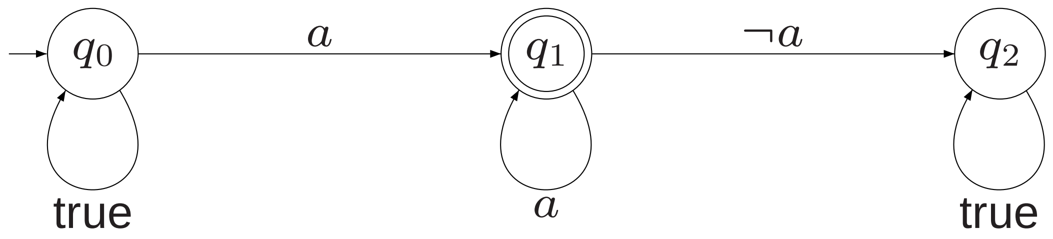
A *persistence property* over *AP* is an LT property $P_{pers} \subseteq \left(2^{AP}\right)^{\omega}$ "eventually for ever Φ" for some propositional logic formula Φ over *AP*:

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

Φ is called a persistence (or state) condition of P_{pers}

"Φ is an invariant after a while"

Example persistence property



let $\{ a \} = AP$, i.e., $2^{AP} = \{A, B\}$ where $A = \{\}$ and $B = \{a\}$

"eventually for ever a" equals $(A + B)^* B^\omega \; = \; (\{\} + \{a\})^* \{a\}^\omega$

Recall synchronous product

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

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

- $S' = S \times Q$, $AP' = Q$ and $L'(\langle s, q \rangle) \;=\; \{ q \}$

- $\rightarrow'$ is the smallest relation defined by: $\dfrac{s \xrightarrow{\alpha} t \;\wedge\; q \xrightarrow{L(t)} p}{\langle s, q \rangle \xrightarrow{\alpha}{}' \langle t, p \rangle}$

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

Verifying ω-regular properties

Let:

- *TS* be a transition system over *AP*
- P be an ω-regular property over *AP*, and
- $\mathcal{A}$ a non-blocking NBA such that $\mathcal{L}_\omega(\mathcal{A}) = \overline{P}$.

The following statements are equivalent:

(a) $\mathit{TS} \models P$

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

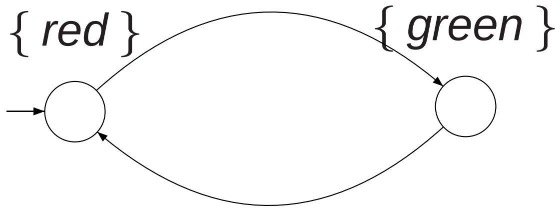
(c) $\mathit{TS} \otimes \mathcal{A} \models P_{pers(A)}$

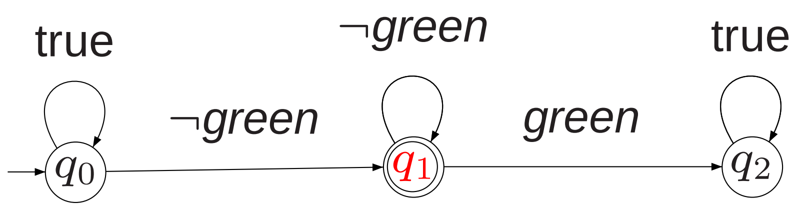
where $P_{pers(A)}$ = *"eventually for ever* $\neg F$*"*

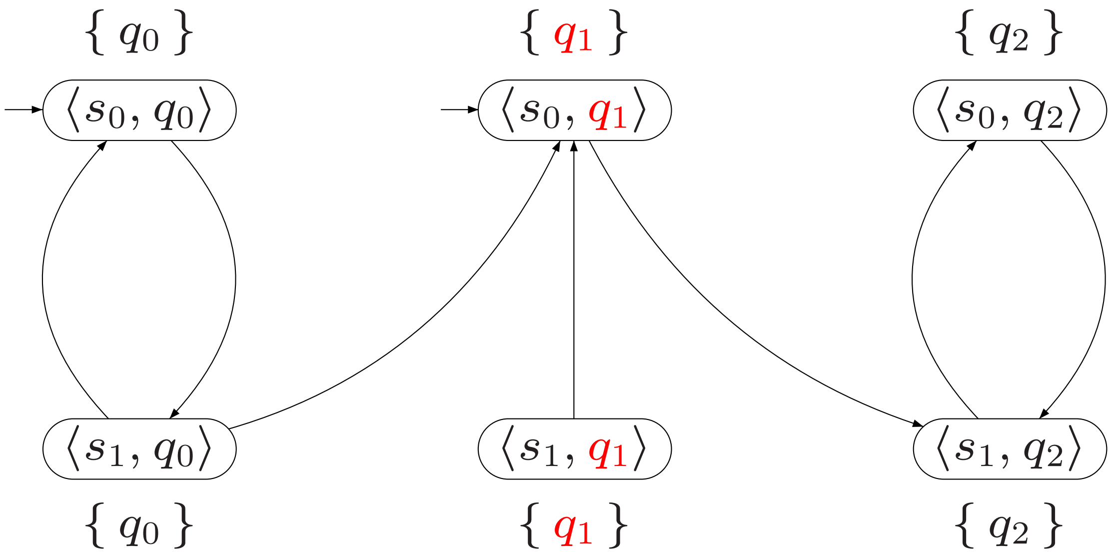
$\Rightarrow$ *checking ω-regular properties is reduced to persistence checking!*

Proof

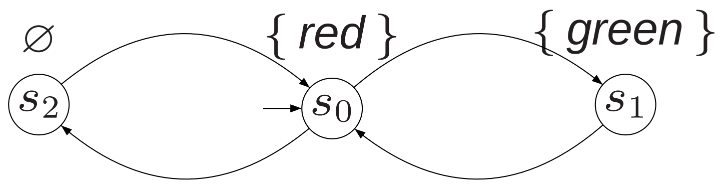
Infinitely often green?

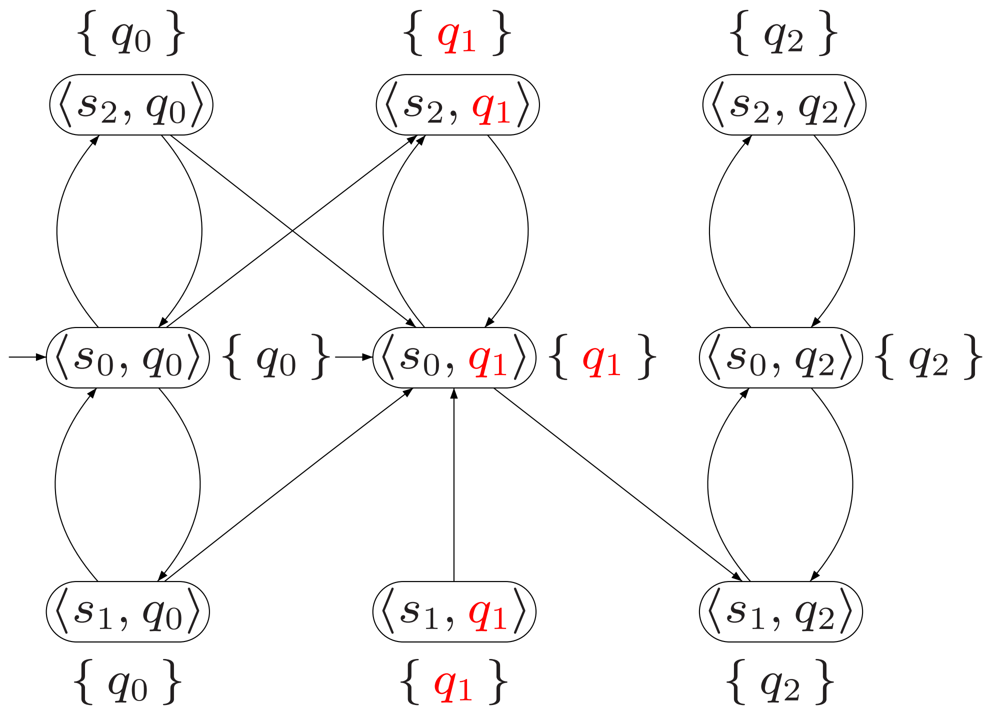






Infinitely often green?





Persistence checking

- Aim: establish whether *TS* $\not\models P_{pers}$ = "eventually for ever Φ"
- Let state s be reachable in *TS* and $s \not\models \Phi$
  - *TS* has an initial path fragment that ends in s
- If s is on a *cycle*
  - this path fragment can be continued by an infinite path
  - . . . . . . by traversing the cycle containing s infinitely often

$\Rightarrow$ *TS* may visit the $\neg\Phi$-state s infinitely often and so: *TS* $\not\models P_{pers}$

- If not such s is found then: *TS* $\models P_{pers}$

In picture

Persistence checking and cycle detection

Let

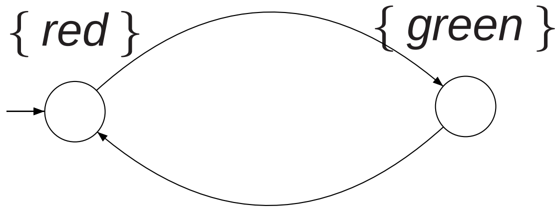
- *TS* be a finite transition system without terminal states over *AP*
- Φ a propositional formula over *AP*, and
- P_{pers} the persistence property ”eventually for ever Φ”

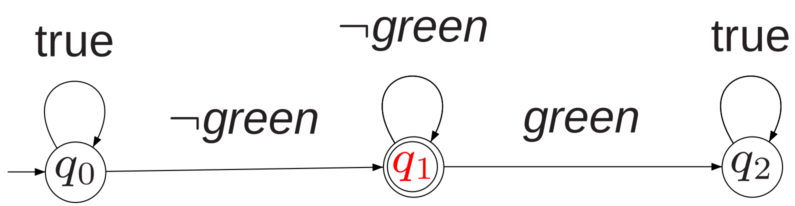
$$TS \not\models P_{pers}$$

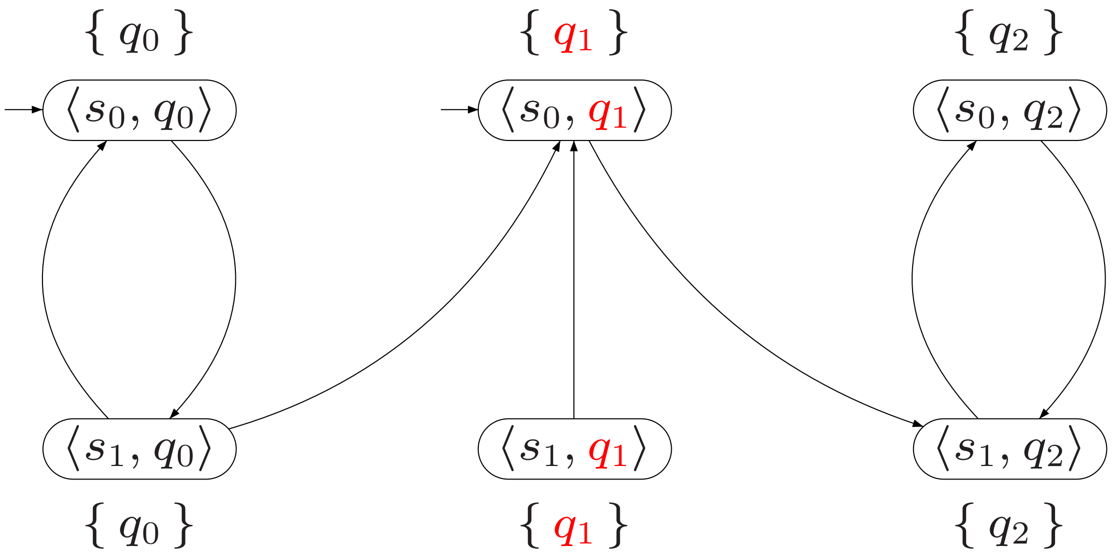
if and only if

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

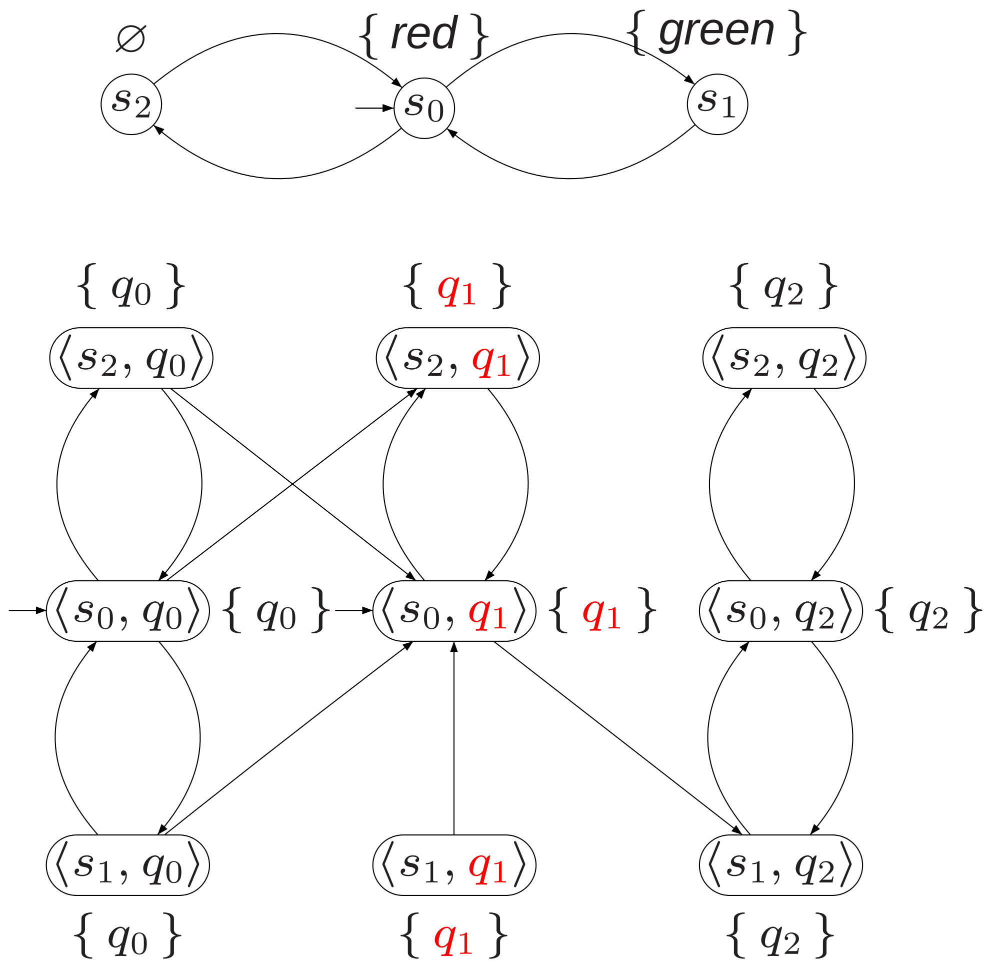
Infinitely often green?







Infinitely often green?



Overview Lecture #11

- Checking Regular Safety Properties
- Checking ω-Regular Properties
  - persistence properties
  - reduction to checking persistence properties
  - checking persistence properties

$\Rightarrow$ Nested Depth-First Search

- Summary of Regular Properties

Cycle detection

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

- Alternative 1:
  - compute the strongly connected components (SCCs) in $G(TS)$
  - check whether one such SCC is reachable from an initial state
  - . . . that contains a $\neg\Phi$-state
  - "eventually for ever Φ" is refuted if and only if such SCC is found

- Alternative 2:
  - *use a nested depth-first search*

  $\Rightarrow$ more adequate for an on-the-fly verification algorithm

  $\Rightarrow$ easier for generating counterexamples

let's have a closer look into this by first dealing with two-phase DFS

A two-phase depth first-search

1. Determine all $\neg\Phi$-states that are reachable from some initial state

   this is performed by a standard depth-first search

2. For each reachable $\neg\Phi$-state, check whether it belongs to a cycle

   - start a depth-first search in s
   - check for all states reachable from s whether there is an "backward" edge to s

- Time complexity: $\Theta(N \cdot |\Phi| \cdot (N+M))$
  - where N is the number of states and M the number of transitions
  - fragments reachable via K $\neg\Phi$-states are searched K times

Two-phase depth first-search

Input: finite transition system *TS* without terminal states, and proposition Φ
Output: "yes" if *TS* $\models$ "eventually for ever Φ", otherwise "no".

---

set of states $R := \varnothing$; $R_{\neg\Phi} := \varnothing$; (* set of reachable states resp. $\neg\Phi$-states *)
stack of states $U := \varepsilon$; (* DFS-stack for first DFS, initial empty *)
set of states $T := \varnothing$; (* set of visited states for the cycle check *)
stack of states $V := \varepsilon$; (* DFS-stack for the cycle check *)

for all $s \in I \setminus R$ **do** visit(s); **od** (* phase one *)
for all $s \in R_{\neg\Phi}$ **do**
  $T := \varnothing$; $V := \varepsilon$; (* phase two *)
  if cycle_check(s) **then** return "no" (* s belongs to a cycle *)
od
return "yes" (* none of the $\neg\Phi$-states belongs to a cycle *)

Find $\neg\Phi$-states

procedure visit (state s)
  push(s, U); (* push s on the stack *)
  $R := R \cup \{ s \}$; (* mark s as reachable *)
  repeat
    $s' :=$ *top*(U);
    if *Post*$(s') \subseteq R$ **then**
      pop(U);
      if $s' \not\models \Phi$ **then** $R_{\neg\Phi} := R_{\neg\Phi} \cup \{ s' \}$; **fi**
    else
      let $s'' \in$ *Post*$(s') \setminus R$
      push(s'', U);
      $R := R \cup \{ s'' \}$; (* state s'' is a new reachable state *)
    fi
  until $(U = \varepsilon)$
endproc

this is a standard DFS checking for $\neg\Phi$-states

Cycle detection

```
procedure boolean  cycle_check(state s)
  boolean  cycle_found := false;                          (* no cycle found yet *)
  push(s, V); T := T ∪ { s };                             (* push s on the stack *)
  repeat
    s' := top(V);                                         (* take top element of V *)
    if s ∈ Post(s') then
      cycle_found := true;                                (* if s ∈ Post(s'), a cycle is found  *)
      push(s, V);                                         (* push s on the stack *)
    else
      if Post(s') \ T ≠ ∅ then
        let s'' ∈ Post(s') \ T;
        push(s'', V); T := T ∪ { s'' };                   (* push an unvisited successor of s' *)
        else  pop(V);                                     (* unsuccessful cycle search for s' *)
      fi
    fi
  until ((V = ε)  ∨  cycle_found)
  return  cycle_found
endproc
```

Nested depth-first search

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

- *Nested DFS*
  - on full expansion of $\neg\Phi$-state s in the outer DFS, start inner DFS
  - in inner DFS, visit all states reachable from s *not visited* in the inner DFS yet
  - no backward edge found to s? continue the outer DFS (look for next $\neg\Phi$ state)

- *Counterexample generation*: DFS stack concatenation
  - stack U for the outer DFS = path fragment from $s_0 \in I$ to s (in reversed order)
  - stack V for the inner DFS = a cycle from state s to s (in reversed order)

The outer DFS (1)

Input: transition system *TS* without terminal states, and proposition Φ
Output: "yes" if *TS* $\models$ "eventually for ever Φ", otherwise "no" plus counterexample

```
set of states R := ∅;                          (* set of visited states in the outer DFS *)
stack of states U := ε;                        (* stack for the outer DFS *)
set of states T := ∅;                          (* set of visited states in the inner DFS *)
stack of states V := ε;                        (* stack for the inner DFS *)
boolean cycle_found := false;

while (I \ R ≠ ∅ ∧ ¬cycle_found) do
   let s ∈ I \ R;                              (* explore the reachable *)
   reachable_cycle(s);                         (* fragment with outer DFS *)
od
if ¬cycle_found then
   return ("yes")                              (* TS |= "eventually for ever Φ" *)
else
   return ("no", reverse(V.U))                 (* stack contents yield a counterexample *)
fi
```

The outer DFS (2)

procedure reachable_cycle (state s)
  push(s, U); (* push s on the stack *)
  $R := R \cup \{ s \}$;
  repeat
    $s' :=$ *top*(U);
    if *Post*$(s') \setminus R \neq \varnothing$ **then**
      let $s'' \in$ *Post*$(s') \setminus R$;
      push(s'', U); (* push the unvisited successor of s' *)
      $R := R \cup \{ s'' \}$; (* and mark it reachable *)
    else
      pop(U); (* outer DFS finished for s' *)
      if $s' \not\models \Phi$ **then**
        cycle_found $:=$ cycle_check(s'); (* proceed with the inner *)
(* DFS in state s' *)
      fi
    fi
  until $((U = \varepsilon) \ \vee$ *cycle_found*$)$ (* stop when stack for the outer *)
(* DFS is empty or cycle found *)
endproc

Example

The order of cycle detection

Correctness of nested DFS

Let:

- *TS* be a finite transition system over *AP* without terminal states and
- P_{pers} a persistence property

The nested DFS algorithm yields "no" if and only if *TS* $\not\models P_{pers}$

Time complexity

The worst-case time complexity of nested DFS is in

$$\mathcal{O}((N+M) + N \cdot |\,\Phi\,|)$$

where N is $\#$ reachable states in *TS*, and M is $\#$ transitions in *TS*

Overview Lecture #11

- Checking Regular Safety Properties
- Checking ω-Regular Properties
  - persistence properties
  - reduction to checking persistence properties
  - checking persistence properties
- Nested Depth-First Search

$\Rightarrow$ Summary of Regular Properties

Summary of regular properties (1)

- Languages recognized by NFA/DFA = regular languages
  - serve to represent the bad prefixes of regular safety properties
- Checking a regular safety property = invariant checking on a product
  - “never visit an accept state” in the NFA for the bad prefixes
  - amounts to solving a (DFS) reachability problem
- ω-regular languages are languages of infinite words
  - can be described by ω-regular expressions
- Languages recognized by NBA = ω-regular languages
  - serve to represent ω-regular properties

Summary of regular properties (2)

- DBA are less powerful than NBA
  - fail, e.g., to represent the persistence property ”eventually for ever a”
- Generalized NBA require repeated visits for several acceptance sets
  - the languages recognized by GNBA = ω-regular languages
- Checking an ω-regular property = checking persistency on a product
  - “eventually for ever no accept state” in the NBA for the complement property
- Persistence checking is solvable in linear time by a nested DFS
- Nested DFS = a DFS for reachable $\neg\Phi$-states + a DFS for cycle detection