Verification

Lecture 6

Bernd Finkbeiner
Peter Faymonville
Michael Gerke

UNIVERSITÄT
DES
SAARLANDES

REVIEW: Büchi automata

A nondeterministic Büchi automaton (NBA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, Q_0, F)$ where:

- Q is a finite set of states with $Q_0 \subseteq Q$ a set of initial states
- Σ is an alphabet
- $\delta : Q \times \Sigma \to 2^Q$ is a transition function
- $F \subseteq Q$ is a set of accept (or: final) states

The size of $\mathcal{A}$, denoted $|\mathcal{A}|$, is the number of states and transitions in $\mathcal{A}$:

$$|\mathcal{A}| \;=\; |Q| + \sum_{q \in Q} \sum_{A \in \Sigma} |\,\delta(q, A)\,|$$

REVIEW: NBA and ω-regular languages

The class of languages accepted by NBA

agrees with the class of ω-regular languages

(1) any ω-regular language is recognized by an NBA

(2) for any NBA $\mathcal{A}$, the language $\mathcal{L}_\omega(\mathcal{A})$ is ω-regular

REVIEW: For any ω-regular language there is an NBA

- How to construct an NBA for the ω-regular expression:

$$G = E_1.F_1^\omega + \ldots + E_n.F_n^\omega \text{ ?}$$

where E_i and F_i are regular expressions over alphabet Σ; $\varepsilon \notin F_i$

- Rely on operations for NBA that mimic operations on ω-regular expressions:
  (1) for NBA $\mathcal{A}_1$ and $\mathcal{A}_2$ there is an NBA accepting $\mathcal{L}_\omega(\mathcal{A}_1) \cup \mathcal{L}_\omega(\mathcal{A}_2)$
  (2) for any regular language $\mathcal{L}$ with $\varepsilon \notin \mathcal{L}$ there is an NBA accepting $\mathcal{L}^\omega$
  (3) for regular language $\mathcal{L}$ and NBA $\mathcal{A}'$ there is an NBA accepting $\mathcal{L}.\mathcal{L}_\omega(\mathcal{A}')$

REVIEW: NBA accept ω-regular languages

For each NBA $\mathcal{A}$: $\mathcal{L}_\omega(\mathcal{A})$ is ω-regular

Proof:

- Given an NBA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$, we define, for each pair $s, s' \in Q$, the regular language $W_{s,s'}$:

$$W_{s,s'} = \{u \in \Sigma^* \mid \text{NFA } (Q, \Sigma, \delta, \{s\}, \{s'\}) \text{ accepts } u\}$$

- $\mathcal{L}_\omega(\mathcal{A}) = \bigcup_{s \in Q_0, s' \in F} W_{s,s'} \,.\, W^\omega_{s',s'}$
- Let $E_{s,s'}$ be the regular expression defining the language $W_{s,s'}$.
- The corresponding ω-regular expression
  $E_{s_1,s_1'}.E^\omega_{s_1',s_1'} \;+\; E_{s_2,s_1'}.E^\omega_{s_1',s_1'} \;+\; \ldots$
  defines $\mathcal{L}_\omega(\mathcal{A})$.

Checking non-emptiness

$\mathcal{L}_\omega(\mathcal{A}) \neq \varnothing$ if and only if

$$\underbrace{\exists q_0 \in Q_0.\ \exists q \in F.\ \exists w \in \Sigma^*.\ \exists v \in \Sigma^+.\ \ q \in \delta^*(q_0, w)\ \wedge\ q \in \delta^*(q, v)}_{\text{there is a reachable accept state on a cycle}}$$

The emptiness problem for NBA $\mathcal{A}$ can be solved in time $\mathcal{O}(|\mathcal{A}|)$

Non-blocking NBA

- NBA $\mathcal{A}$ is non-blocking if $\delta(q, A) \neq \varnothing$ for all q and $A \in \Sigma$
  - for each input word there exists an infinite run
- For each NBA $\mathcal{A}$ there exists a non-blocking NBA $\mathit{trap}(\mathcal{A})$ with:
  - $|\mathit{trap}(\mathcal{A})| = \mathcal{O}(|\mathcal{A}|)$ and $\mathcal{A} \equiv \mathit{trap}(\mathcal{A})$
- For $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ let $\mathit{trap}(\mathcal{A}) = (Q', \Sigma, \delta', Q_0, F)$ with:
  - $Q' = Q \cup \{ q_{trap} \}$ where $\{ q_{trap} \} \notin Q$
  - $\delta'(q, A) = \begin{cases} \delta(q, A) & : \text{ if } q \in Q \text{ and } \delta(q, A) \neq \varnothing \\ \{q_{trap}\} & : \text{ otherwise} \end{cases}$

Deterministic BA

Büchi automaton $\mathcal{A}$ is called deterministic if

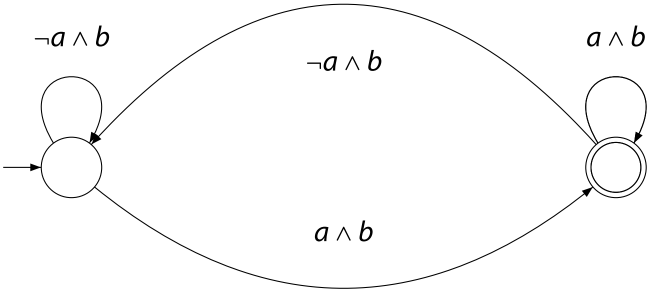
$$|Q_0| \leq 1 \quad \text{and} \quad |\delta(q, A)| \leq 1 \quad \text{for all } q \in Q \text{ and } A \in \Sigma$$

DBA $\mathcal{A}$ is called total if

$$|Q_0| = 1 \quad \text{and} \quad |\delta(q, A)| = 1 \quad \text{for all } q \in Q \text{ and } A \in \Sigma$$

total DBA provide unique runs for each input word

Example DBA for LT property



NBA are more expressive than DBA

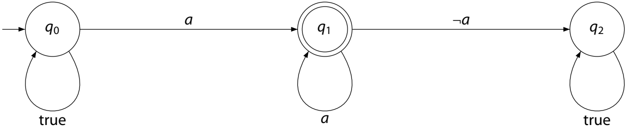
NFA and DFA are equally expressive but NBA and DBA are not!

There is no DBA that accepts $\mathcal{L}_\omega((A + B)^* B^\omega)$

Proof

- Assume that $L = \mathcal{L}((A + B)^* B^\omega)$ is recognized by the deterministic Büchi automaton $\mathcal{A}$.
- Since $b^\omega \in L$, there is a run
  $r_0 = s_{0,0} s_{0,1} s_{0,2}, \ldots$
  with $s_{0,n_0} \in F$ for some $n_0 \in \mathbb{N}$.
- Similarly, $b^{n_0} a b^\omega \in L$ and there must be a run
  $r_1 = s_{0,0} s_{0,1} s_{0,2} \ldots s_{0,n_0} s_{1,0} s_{1,1} s_{1,2} \ldots$
  with $s_{1,n_1} \in F$
- Repeating this argument, there is a word
  $b^{n_0} a b^{n_1} a b^{n_2} a \ldots$
  accepted by $\mathcal{A}$.
- This contradicts $L = \mathcal{L}_\omega(\mathcal{A})$.

The need for nondeterminism



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

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

Generalized Büchi automata

- NBA are as expressive as ω-regular languages
- Variants of NBA exist that are equally expressive
  - Muller, Rabin, and Streett automata
  - generalized Büchi automata (GNBA)
- GNBA are like NBA, but have a distinct acceptance criterion
  - a GNBA requires to visit several sets $F_1, \ldots, F_k$ ($k \geq 0$) infinitely often
  - for k=0, all runs are accepting
  - for k=1 this boils down to an NBA
- GNBA are useful to relate temporal logic and automata
  - but they are equally expressive as NBA

Generalized Büchi automata

A generalized NBA (GNBA) $\mathcal{G}$ is a tuple $(Q, \Sigma, \delta, Q_0, \mathcal{F})$ where:

- Q is a finite set of states with $Q_0 \subseteq Q$ a set of initial states
- Σ is an alphabet
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function
- $\mathcal{F} = \{ F_1, \ldots, F_k \}$ is a (possibly empty) subset of 2^Q

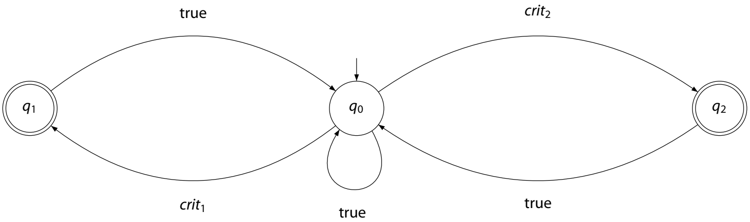
The size of $\mathcal{G}$, denoted $|\mathcal{G}|$, is the number of states and transitions in $\mathcal{G}$:

$$|\mathcal{G}| = |Q| + \sum_{q \in Q} \sum_{A \in \Sigma} |\delta(q, A)|$$

Language of a GNBA

- GNBA $\mathcal{G} = (Q, \Sigma, \delta, Q_0, \mathcal{F})$ and word $\sigma = A_0 A_1 A_2 \ldots \in \Sigma^\omega$
- A *run* for σ in $\mathcal{G}$ is an infinite sequence $q_0\, q_1\, q_2 \ldots$ such that:
  - $q_0 \in Q_0$ and $q_i \xrightarrow{A_i} q_{i+1}$ for all $0 \leq i$
- Run $q_0\, q_1 \ \ldots$ is accepting if for all $F \in \mathcal{F}$: $q_i \in F$ for infinitely many i
- $\sigma \in \Sigma^\omega$ is *accepted* by $\mathcal{G}$ if there exists an accepting run for σ
- The accepted language of $\mathcal{G}$:
  - $\mathcal{L}_\omega(\mathcal{G}) = \left\{ \sigma \in \Sigma^\omega \mid \text{there exists an accepting run for } \sigma \text{ in } \mathcal{G} \right\}$
- GNBA $\mathcal{G}$ and $\mathcal{G}'$ are equivalent if $\mathcal{L}_\omega(\mathcal{G}) = \mathcal{L}_\omega(\mathcal{G}')$

Example



A GNBA for the property "both processes are infinitely often in their critical section"

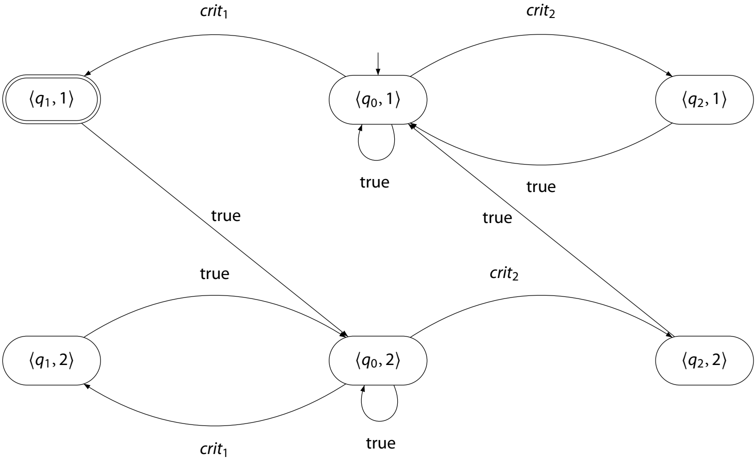
From GNBA to NBA

For any GNBA $\mathcal{G}$ there exists an NBA $\mathcal{A}$ with:

$$\mathcal{L}_\omega(\mathcal{G}) = \mathcal{L}_\omega(\mathcal{A}) \text{ and } |\mathcal{A}| = \mathcal{O}(|\mathcal{G}| \cdot |\mathcal{F}|)$$

where $\mathcal{F}$ denotes the set of acceptance sets in $\mathcal{G}$

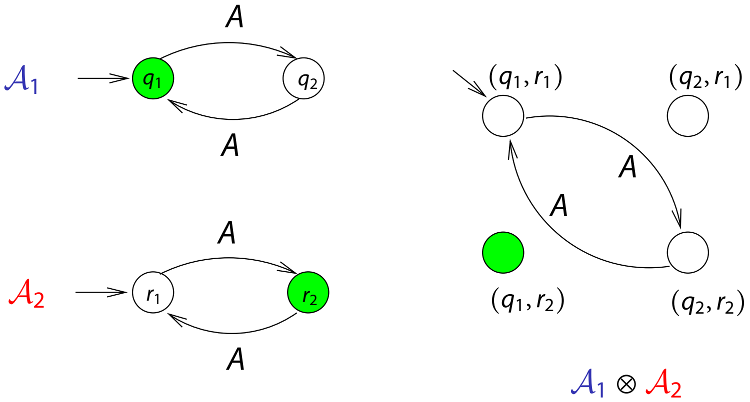
Example



crit$_1$
crit$_2$
$\langle q_1, 1\rangle$
$\langle q_0, 1\rangle$
$\langle q_2, 1\rangle$
true
true
true
true
true
crit$_2$
$\langle q_1, 2\rangle$
$\langle q_0, 2\rangle$
$\langle q_2, 2\rangle$
crit$_1$
true

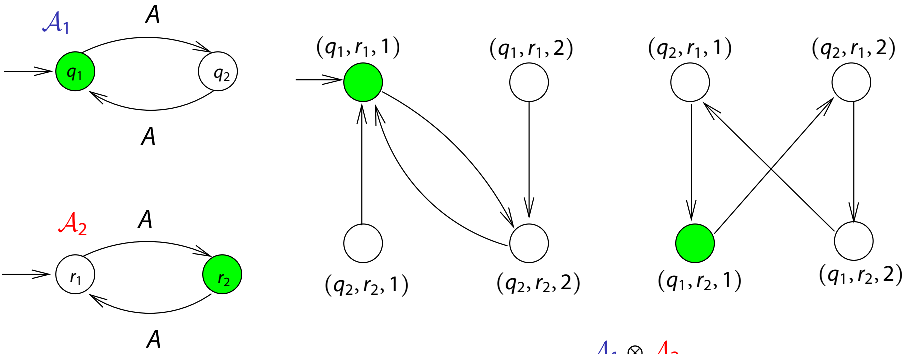
Product of Büchi automata

The product construction for finite automata does not work:



$\mathcal{L}_\omega(\mathcal{A}_1) = \mathcal{L}_\omega(\mathcal{A}_2) = \{ A^\omega \}$, but $\mathcal{L}_\omega(\mathcal{A}_1 \otimes \mathcal{A}_2) = \varnothing$

Product of Büchi automata



$\mathcal{A}_1 \otimes \mathcal{A}_2$

Intersection

For GNBA $\mathcal{G}_1$ and $\mathcal{G}_2$ there exists a GNBA $\mathcal{G}$ with

$$\mathcal{L}_\omega(\mathcal{G}) = \mathcal{L}_\omega(\mathcal{G}_1) \cap \mathcal{L}_\omega(\mathcal{G}_2) \quad \text{and} \quad |\mathcal{G}| = \mathcal{O}(|\mathcal{G}_1| + |\mathcal{G}_2|)$$

Facts about Büchi automata

- They are as expressive as ω-regular languages
- They are closed under various operations and also under $\cap$
  - deterministic automaton $-\mathcal{A}$ accepts $-\mathcal{L}_\omega(\mathcal{A})$
- Nondeterministic BA are more expressive than deterministic BA
- Emptiness check = check for reachable recurrent accept state
  - this can be done in $\mathcal{O}(|\mathcal{A}|)$

Verifying ω-regular properties

REVIEW: Regular safety properties

Safety property P_{safe} over AP is regular

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

REVIEW: 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" $\neg F$

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

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

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

Persistence property

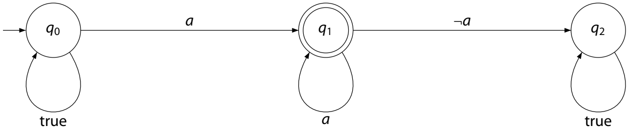
A persistence property over *AP* is an LT property $P_{pers} \subseteq \left(2^{AP}\right)^{\omega}$
"eventually forever Φ"
for some propositional logic formula Φ over *AP*:

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

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

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) $TS \models P$

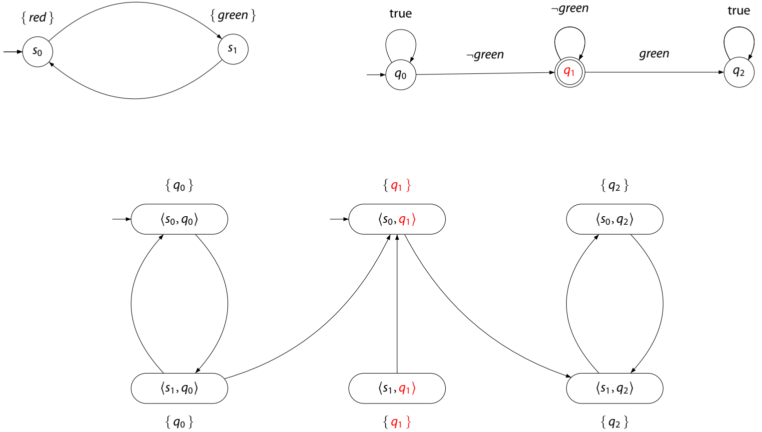
(b) $Traces(TS) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

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

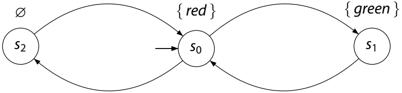
where $P_{pers(A)}$ = "eventually forever $\neg F$"

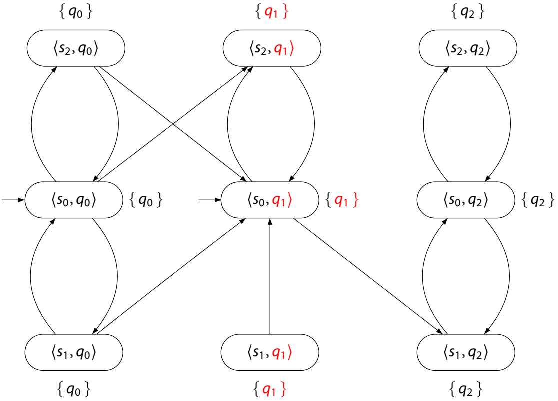
$\Rightarrow$ checking ω-regular properties is reduced to persistence checking!

Infinitely often green?



Infinitely often green?





Persistence checking

- Aim: establish whether $TS \not\models P_{pers}$ = "eventually forever 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 no such s is found then: $TS \models P_{pers}$

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 forever 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

Require: finite transition system *TS* without terminal states, and proposition Φ
Ensure: "yes" if $TS \models$ "eventually forever 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}
end for
return "yes" {none of the $\neg\Phi$-states belongs to a cycle}

Find $\neg\Phi$-states

```
process visit (state s)
push(s, U); {push s on the stack}
R := R ∪ { s }; {mark s as reachable}
repeat
    s' := top(U);
    if Post(s') ⊆ R then
        pop(U);
        if s' ⊭ Φ then R_¬Φ := R_¬Φ ∪ { s' }; fi
    else
        let s'' ∈ Post(s') \ R
        push(s'', U);
        R := R ∪ { s'' }; {state s'' is a new reachable state}
    end if
until (U = ε) endproc
```

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

Cycle detection

process boolean cycle_check(state s)
  boolean *cycle_found* := false; {no cycle found yet}
  $push(s, V)$; $T := T \cup \{ s \}$; {push s on the stack}
  repeat
    $s' := top(V)$; {take top element of V}
    if $s \in Post(s')$ **then**
      cycle_found := true; {if $s \in Post(s')$, a cycle is found }
      $push(s, V)$; {push s on the stack}
    else
      if $Post(s') \setminus T \neq \varnothing$ **then**
        let $s'' \in Post(s') \setminus T$;
        $push(s'', V)$; $T := T \cup \{ s'' \}$; {push an unvisited successor of s'}
        else $pop(V)$; {unsuccessful cycle search for s'}
      end if
    end if
  until $((V = \varepsilon) \vee cycle\_found)$
  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 the $\neg\Phi$-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)

Require: transition system *TS* without terminal states, and proposition Φ
Ensure: "yes" if $TS \models$ "eventually forever Φ", otherwise "no" plus counterexample

---

set of states $R := \varnothing$; {set of visited states in the outer DFS}
stack of states $U := \varepsilon$; {stack for the outer DFS}
set of states $T := \varnothing$; {set of visited states in the inner DFS}
stack of states $V := \varepsilon$; {stack for the inner DFS}
boolean *cycle_found* := false;

while $(I \setminus R \neq \varnothing \ \wedge \ \neg cycle\_found)$ **do**
  let $s \in I \setminus R$; {explore the reachable}
  reachable_cycle(*s*); {fragment with outer DFS}
end while
if $\neg cycle\_found$ **then**
  return ("yes") {$TS \models$ "eventually forever Φ"}
else
  return ("no", *reverse*$(V.U)$) {stack contents yield a counterexample}
end if

The outer DFS (2)

```
process reachable_cycle (state s)
push(s, U); {push s on the stack}
R := R ∪ { s };
repeat
    s' := top(U);
    if Post(s') ∖ R ≠ ∅ then
        let s'' ∈ Post(s') ∖ R;
        push(s'', U); {push the unvisited successor of s'}
        R := R ∪ { s'' }; {and mark it reachable}
    else
        pop(U); {outer DFS finished for s'}
        if s' ⊭ Φ then
            cycle_found := cycle_check(s'); {proceed with the inner}
            {DFS in state s'}
        end if
    end if
until ((U = ε) ∨ cycle_found) {stop when stack for the outer}
{DFS is empty or cycle found} endproc
```

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

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

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

Linear-time Temporal Logic

Syntax

modal logic over infinite sequences [Pnueli 1977]

- Propositional logic
  - $a \in AP$ — atomic proposition
  - $\neg\phi$ and $\phi \wedge \psi$ — negation and conjunction
- Temporal operators
  - $\bigcirc \phi$ — next state fulfills ϕ
  - $\phi \, \mathsf{U} \, \psi$ — ϕ holds Until a ψ-state is reached

linear temporal logic is a logic for describing LT properties

Derived operators

$$\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$$

$$\phi \Rightarrow \psi \equiv \neg\phi \vee \psi$$

$$\phi \Leftrightarrow \psi \equiv (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$$

$$\phi \oplus \psi \equiv (\phi \wedge \neg\psi) \vee (\neg\phi \wedge \psi)$$

$$\text{true} \equiv \phi \vee \neg\phi$$

$$\text{false} \equiv \neg\,\text{true}$$

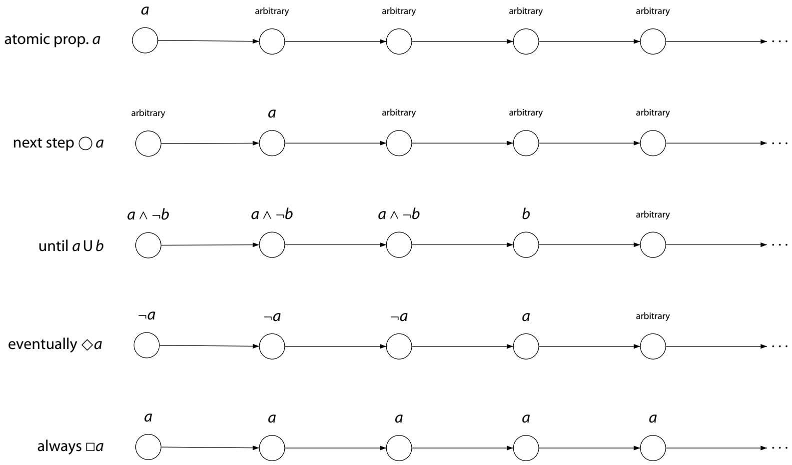
$$\Diamond\phi \equiv \text{true}\ \mathsf{U}\ \phi \qquad \text{“sometimes in the future”}$$

$$\Box\phi \equiv \neg\Diamond\neg\phi \qquad \text{“from now on forever”}$$

precedence order: the unary operators bind stronger than the binary ones.

$\neg$ and $\bigcirc$ bind equally strong. U takes precedence over $\wedge$, $\vee$, and $\rightarrow$

Intuitive semantics



Traffic light properties

- Once red, the light cannot become green immediately:

$$\square\,(\mathit{red} \;\Rightarrow\; \neg \bigcirc \mathit{green})$$

- The light becomes green eventually: $\Diamond\, \mathit{green}$
- Once red, the light always becomes green eventually: $\square\,(\mathit{red} \;\Rightarrow\; \Diamond\, \mathit{green})$
- Once red, the light always becomes green eventually after being yellow for some time inbetween:

$$\square(\mathit{red} \rightarrow \bigcirc\,(\mathit{red}\,\mathrm{U}\,(\mathit{yellow} \;\wedge\; \bigcirc\,(\mathit{yellow}\,\mathrm{U}\,\mathit{green}))))$$

Semantics over words

The LT-property induced by LTL formula φ over AP is:

$$Words(\varphi) = \left\{\sigma \in \left(2^{AP}\right)^{\omega} \mid \sigma \models \varphi\right\},$$ where $\models$ is the smallest relation satisfying:

$$
\begin{array}{lllll}
\sigma & \models & \text{true} & & \\
\sigma & \models & a & \text{iff} & a \in A_0 \quad (\text{i.e., } A_0 \models a) \\
\sigma & \models & \varphi_1 \wedge \varphi_2 & \text{iff} & \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma & \models & \neg\,\varphi & \text{iff} & \sigma \not\models \varphi \\
\sigma & \models & \bigcirc\,\varphi & \text{iff} & \sigma[1..] = A_1 A_2 A_3 \ldots \models \varphi \\
\sigma & \models & \varphi_1 \,\mathsf{U}\, \varphi_2 & \text{iff} & \exists j \geq 0.\ \sigma[j..] \models \varphi_2 \text{ and } \sigma[i..] \models \varphi_1,\ 0 \leq i < j
\end{array}
$$

for $\sigma = A_0 A_1 A_2 \ldots$ we have $\sigma[i..] = A_i A_{i+1} A_{i+2} \ldots$ is the suffix of σ from index i on

Semantics over paths and states

Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system without terminal states, and let φ be an LTL-formula over AP.

- For infinite path fragment π of TS:

$$\pi \models \varphi \qquad \text{iff} \qquad trace(\pi) \models \varphi$$

- For state $s \in S$:

$$s \models \varphi \qquad \text{iff} \qquad (\forall \pi \in Paths(s).\ \pi \models \varphi)$$

- TS satisfies φ, denoted $TS \models \varphi$, if $Traces(TS) \subseteq Words(\varphi)$

Semantics for transition systems

$$TS \models \varphi$$

iff (* transition system semantics *)

$$Traces(TS) \subseteq Words(\varphi)$$

iff (* definition of $\models$ for LT-properties *)

$$TS \models Words(\varphi)$$

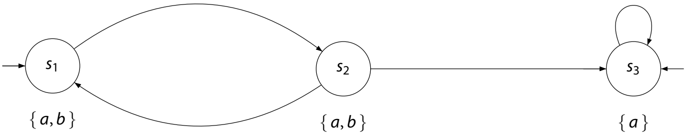
iff (* Definition of $Words(\varphi)$ *)

$$\pi \models \varphi \text{ for all } \pi \in Paths(TS)$$

iff (* semantics of $\models$ for states *)

$$s_0 \models \varphi \text{ for all } s_0 \in I \quad .$$

Example



Semantics of negation

For paths, it holds $\pi \models \varphi$ if and only if $\pi \not\models \neg\varphi$ since:

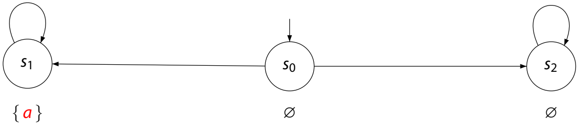
$$Words(\neg\varphi) = \left(2^{AP}\right)^{\omega} \setminus Words(\varphi) \quad .$$

But: $TS \not\models \varphi$ and $TS \models \neg\varphi$ are not equivalent in general
It holds: $TS \models \neg\varphi$ implies $TS \not\models \varphi$. Not always the reverse!
Note that:

$$\begin{aligned} TS \not\models \varphi \quad & \text{iff} \quad Traces(TS) \not\subseteq Words(\varphi) \\ & \text{iff} \quad Traces(TS) \setminus Words(\varphi) \neq \varnothing \\ & \text{iff} \quad Traces(TS) \cap Words(\neg\varphi) \neq \varnothing \quad . \end{aligned}$$

TS neither satisfies φ nor $\neg\varphi$ if there are paths π_1 and π_2 in *TS* such that $\pi_1 \models \varphi$ and $\pi_2 \models \neg\varphi$

Example



A transition system for which $TS \not\models \Diamond a$ and $TS \not\models \neg \Diamond a$