Model Checking I
alias
Reactive Systems Verification

Luca Tesei

MSc in Computer Science, University of Camerino

Topics

- Safety and Liveness.
- Recall of Propositional Logic.
- Invariant properties. Examples.
- Invariant properties checking via Depth-First Search.

Material

Reading:

Chapter 3 of the book, pages 106–111.

More:

The slides in the following pages are taken from the material of the course “Introduction to Model Checking” held by Prof. Dr. Ir. Joost-Pieter Katoen at Aachen University.

Overview

Classification of LT-properties

safety properties *"nothing bad will happen"*

liveness properties *"something good will happen"*

Classification of LT-properties

safety properties *"nothing bad will happen"*

examples:

- mutual exclusion
- deadlock freedom
- "every red phase is preceded by a yellow phase"

liveness properties *"something good will happen"*

Classification of LT-properties

safety properties *"nothing bad will happen"*

examples:

- mutual exclusion
- deadlock freedom
- "every red phase is preceded by a yellow phase"

liveness properties *"something good will happen"*

examples:

- "each waiting process will eventually enter its critical section"
- "each philosopher will eat infinitely often"

Classification of LT-properties

safety properties *"nothing bad will happen"*

examples:

- mutual exclusion
- deadlock freedom

} special case: **invariants** *"no bad state will be reached"*

- "every red phase is preceded by a yellow phase"

liveness properties *"something good will happen"*

examples:

- "each waiting process will eventually enter its critical section"
- "each philosopher will eat infinitely often"

Propositional logic

$$\Phi ::= \mathit{true} \;\Big|\; a \;\Big|\; \Phi_1 \wedge \Phi_2 \;\Big|\; \neg\Phi$$

Propositional logic

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi$$

atomic proposition, i.e., $a \in AP$

Propositional logic

IS2.5-2

$$\Phi ::= \textit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \rightarrow \Phi_2 \mid \ldots$$

atomic proposition, i.e., $a \in AP$

Propositional logic

$$\Phi \ ::= \ \textit{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \Phi_1 \vee \Phi_2 \ \Big| \ \Phi_1 \rightarrow \Phi_2 \ \Big| \ \ldots$$

atomic proposition, i.e., $a \in AP$

semantics: interpretation over a subsets of AP

Propositional logic

$$\Phi ::= \textit{true} \;\Big|\; a \;\Big|\; \Phi_1 \wedge \Phi_2 \;\Big|\; \neg\Phi \;\Big|\; \Phi_1 \vee \Phi_2 \;\Big|\; \Phi_1 \rightarrow \Phi_2 \;\Big|\; \ldots$$

↑ atomic proposition, i.e., $a \in AP$

semantics: Let $A \subseteq AP$

$$
\begin{array}{lll}
A \models \textit{true} & & \\
A \models a & \text{iff} & a \in A \\
A \models \Phi_1 \wedge \Phi_2 & \text{iff} & A \models \Phi_1 \text{ and } A \models \Phi_2 \\
A \models \neg\Phi & \text{iff} & A \not\models \Phi
\end{array}
$$

Propositional logic

$$\Phi \ ::= \ \textit{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \Phi_1 \vee \Phi_2 \ \Big| \ \Phi_1 \rightarrow \Phi_2 \ \Big| \ \ldots$$

↑ atomic proposition, i.e., $a \in AP$

semantics: Let $A \subseteq AP$

$$\begin{array}{lll} A \models \textit{true} & & \\ A \models a & \text{iff} & a \in A \\ A \models \Phi_1 \wedge \Phi_2 & \text{iff} & A \models \Phi_1 \text{ and } A \models \Phi_2 \\ A \models \neg\Phi & \text{iff} & A \not\models \Phi \end{array}$$

e.g., $\{a, b\} \not\models (a \rightarrow \neg b) \vee c \qquad \{a, b\} \models a \vee c$

Propositional logic

$$\Phi \ ::= \ \textit{true} \ \big| \ a \ \big| \ \Phi_1 \wedge \Phi_2 \ \big| \ \neg\Phi \ \big| \ \Phi_1 \vee \Phi_2 \ \big| \ \Phi_1 \rightarrow \Phi_2 \ \big| \ \ldots$$

↑ atomic proposition, i.e., $a \in AP$

semantics: Let $A \subseteq AP$

$A \models \textit{true}$		
$A \models a$	iff	$a \in A$
$A \models \Phi_1 \wedge \Phi_2$	iff	$A \models \Phi_1$ and $A \models \Phi_2$
$A \models \neg\Phi$	iff	$A \not\models \Phi$

for state s of a TS over AP: $s \models \Phi$ iff $L(s) \models \Phi$

Invariant

Invariant

Let E be an LT property over AP.

E is called an invariant if there exists a propositional formula Φ over AP such that

$$E = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} : \forall i \geq 0.\, A_i \models \Phi \right\}$$

Invariant

Let E be an LT property over AP.

E is called an invariant if there exists a propositional formula Φ over AP such that

$$E = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} : \forall i \geq 0.\, A_i \models \Phi \right\}$$

Φ is called the invariant condition of E.

Examples for invariants

mutual exclusion (safety):

$$MUTEX = \begin{array}{l} \text{set of all infinite words } A_0\,A_1\,A_2\ldots \text{ s.t.} \\ \forall i \in \mathbb{N}.\ \ \text{crit}_1 \notin A_i \ \text{ or } \ \text{crit}_2 \notin A_i \end{array}$$

here: $AP = \{\text{crit}_1, \text{crit}_2, \ldots\}$

Examples for invariants

mutual exclusion (safety):

$$MUTEX = \begin{array}{c}\text{set of all infinite words } A_0\, A_1\, A_2 \ldots \text{ s.t.} \\ \forall i \in \mathbb{N}.\ \ \mathsf{crit}_1 \notin A_i \ \text{ or } \ \mathsf{crit}_2 \notin A_i\end{array}$$

invariant condition: $\Phi = \neg \mathsf{crit}_1 \vee \neg \mathsf{crit}_2$

here: $AP = \{\mathsf{crit}_1, \mathsf{crit}_2, \ldots\}$

Examples for invariants

mutual exclusion (safety):

$$MUTEX = \text{set of all infinite words } A_0 A_1 A_2 \ldots \text{ s.t. } \forall i \in \mathbb{N}.\ \text{crit}_1 \notin A_i \text{ or } \text{crit}_2 \notin A_i$$

invariant condition: $\Phi = \neg\text{crit}_1 \vee \neg\text{crit}_2$

deadlock freedom for 5 dining philosophers:

$$DF = \text{set of all infinite words } A_0 A_1 A_2 \ldots \text{ s.t. } \forall i \in \mathbb{N}\ \exists j \in \{0,1,2,3,4\}.\ \text{wait}_j \notin A_i$$

here: $AP = \{\text{wait}_j : 0 \leq j \leq 4\} \cup \{\ldots\}$

Examples for invariants

mutual exclusion (safety):

$$MUTEX = \begin{array}{c}\text{set of all infinite words } A_0\, A_1\, A_2 \ldots \text{ s.t.}\\ \forall i \in \mathbb{N}.\ \ \mathsf{crit}_1 \notin A_i \ \text{ or } \ \mathsf{crit}_2 \notin A_i\end{array}$$

invariant condition: $\Phi = \neg\mathsf{crit}_1 \vee \neg\mathsf{crit}_2$

deadlock freedom for 5 dining philosophers:

$$DF = \begin{array}{c}\text{set of all infinite words } A_0\, A_1\, A_2 \ldots \text{ s.t.}\\ \forall i \in \mathbb{N}\ \exists \mathrm{j} \in \{0,1,2,3,4\}.\ \mathsf{wait}_j \notin A_i\end{array}$$

invariant condition: **?**

here: $AP = \{\mathsf{wait}_j : 0 \leq j \leq 4\} \cup \{\ldots\}$

Examples for invariants

mutual exclusion (safety):

$$MUTEX = \begin{array}{l} \text{set of all infinite words } A_0\,A_1\,A_2 \ldots \text{ s.t.} \\ \forall i \in \mathbb{N}.\ \ \mathsf{crit}_1 \notin A_i \ \text{ or } \ \mathsf{crit}_2 \notin A_i \end{array}$$

invariant condition: $\Phi = \neg\mathsf{crit}_1 \vee \neg\mathsf{crit}_2$

deadlock freedom for 5 dining philosophers:

$$DF = \begin{array}{l} \text{set of all infinite words } A_0\,A_1\,A_2 \ldots \text{ s.t.} \\ \forall i \in \mathbb{N}\ \exists \mathbf{j} \in \{0,1,2,3,4\}.\ \mathsf{wait}_j \notin A_i \end{array}$$

invariant condition:

$$\Phi = \neg\mathsf{wait}_0 \vee \neg\mathsf{wait}_1 \vee \neg\mathsf{wait}_2 \vee \neg\mathsf{wait}_3 \vee \neg\mathsf{wait}_4$$

here: $AP = \{\mathsf{wait}_j : 0 \leq j \leq 4\} \cup \{\ldots\}$

Satisfaction of invariants

Let E be an LT property over AP. E is called an invariant if there exists a propositional formula Φ s.t.

$$E = \{ A_0 A_1 A_2 \ldots \in (2^{AP})^\omega : \forall i \geq 0. A_i \models \Phi \}$$

Let E be an LT property over AP. E is called an invariant if there exists a propositional formula Φ s.t.

$$E = \{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^\omega : \forall i \geq 0.\, A_i \models \Phi \}$$

Let $\mathcal{T}$ be a TS over AP without terminal states. Then:

$\mathcal{T} \models E$ iff $trace(\pi) \in E$ for all $\pi \in Paths(\mathcal{T})$

Satisfaction of invariants

Let E be an LT property over AP. E is called an invariant if there exists a propositional formula Φ s.t.

$$E = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} : \forall i \geq 0.\, A_i \models \Phi \right\}$$

Let $\mathcal{T}$ be a TS over AP without terminal states. Then:

$\mathcal{T} \models E$ iff $trace(\pi) \in E$ for all $\pi \in Paths(\mathcal{T})$

iff $s \models \Phi$ for all states s on a path of $\mathcal{T}$

Satisfaction of invariants

Let E be an LT property over AP. E is called an invariant if there exists a propositional formula Φ s.t.

$$E = \{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^\omega : \forall i \geq 0.\ A_i \models \Phi \}$$

Let $\mathcal{T}$ be a TS over AP without terminal states. Then:

$\mathcal{T} \models E$ iff $trace(\pi) \in E$ for all $\pi \in Paths(\mathcal{T})$

iff $s \models \Phi$ for all states s on a path of $\mathcal{T}$

iff $s \models \Phi$ for all states $s \in Reach(\mathcal{T})$

↑ set of reachable states in $\mathcal{T}$

Satisfaction of invariants

Let E be an LT property over AP. E is called an invariant if there exists a propositional formula Φ s.t.

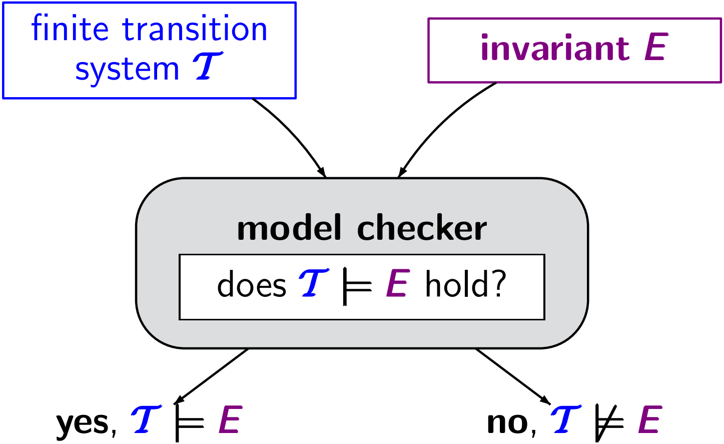
$$E = \{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^\omega : \forall i \geq 0.\, A_i \models \Phi \}$$

Let $\mathcal{T}$ be a TS over AP without terminal states. Then:

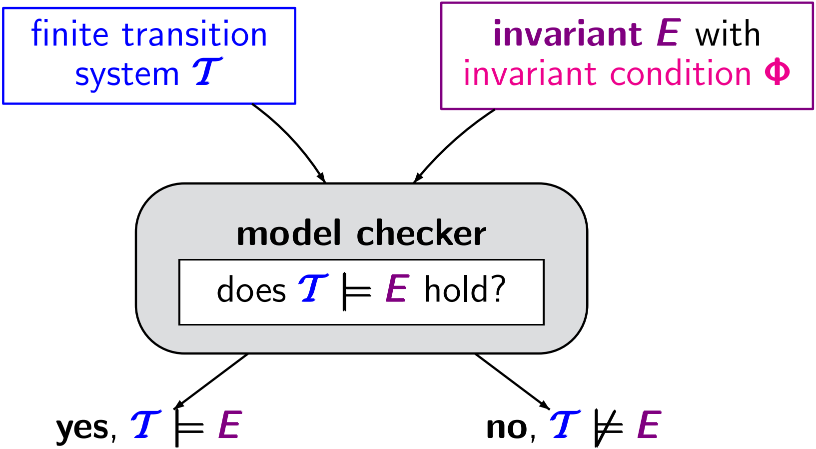
$$\begin{aligned} \mathcal{T} \models E \quad & \text{iff} \quad \mathit{trace}(\pi) \in E \text{ for all } \pi \in \mathit{Paths}(\mathcal{T}) \\ & \text{iff} \quad s \models \Phi \text{ for all states } s \text{ on a path of } \mathcal{T} \\ & \text{iff} \quad s \models \Phi \text{ for all states } s \in \mathit{Reach}(\mathcal{T}) \end{aligned}$$

i.e., Φ holds in all initial states and is invariant under all transitions

Invariant checking

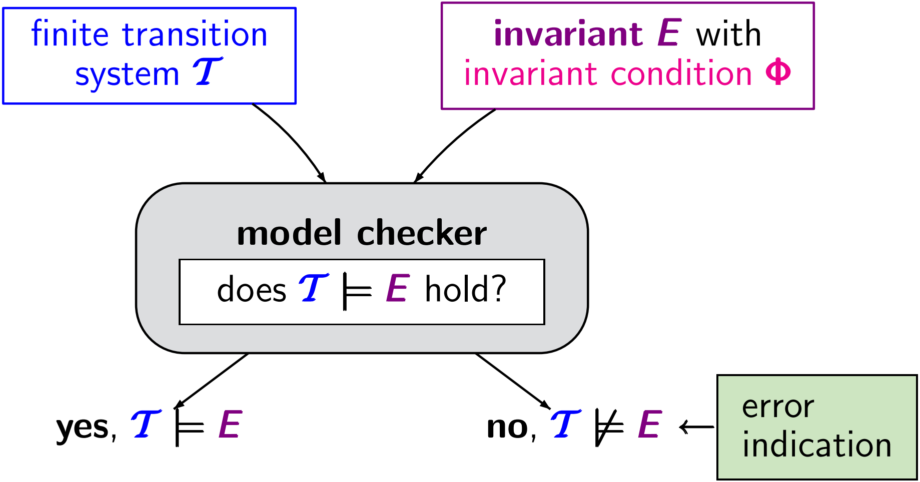


Invariant checking



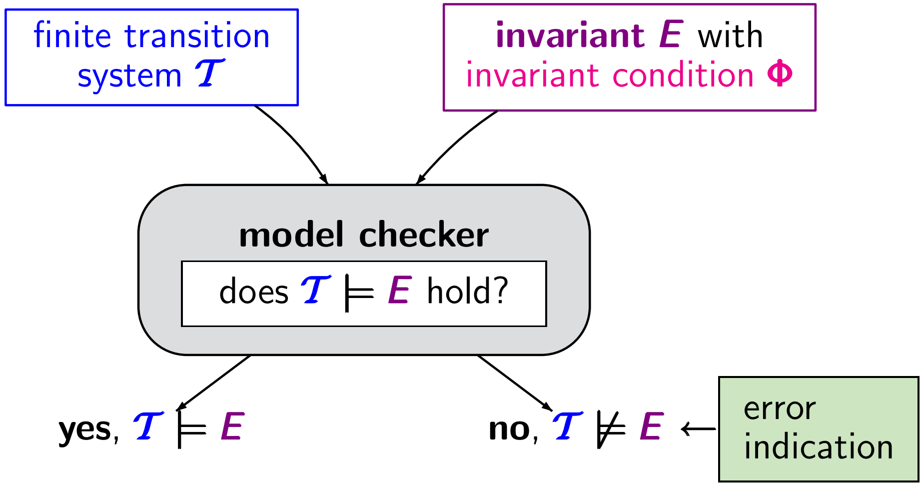
perform a graph analysis (**DFS** or **BFS**) to check whether $s \models \Phi$ for all $s \in Reach(\mathcal{T})$

Invariant checking



perform a graph analysis (**DFS** or **BFS**) to check whether $s \models \Phi$ for all $s \in \mathit{Reach}(\mathcal{T})$

Invariant checking



error indication: initial path fragment $s_0\, s_1 \ldots s_{n-1} s_n$
such that $s_i \models \Phi$ for $0 \leq i < n$ and $s_n \not\models \Phi$

DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

```
FOR ALL s_0 ∈ S_0 DO
      IF DFS(s_0, Φ) THEN
         return "no"
      FI
OD
return "yes"
```

DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

```
FOR ALL s0 ∈ S0 DO
     IF DFS(s0, Φ) THEN
          return "no"
     FI
OD
return "yes"
```

$DFS(s_0, \Phi)$ returns "true" iff depth-first search from state s_0 leads to some state t with $t \not\models \Phi$

DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

```
π := ∅  ⟵  stack for error indication
FOR ALL s0 ∈ S0 DO
      IF DFS(s0, Φ) THEN
            return "no" and reverse(π)
      FI
OD
return "yes"
```

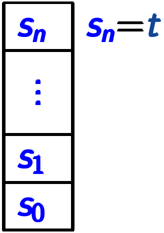
$DFS(s_0, \Phi)$ returns "true" iff depth-first search from state s_0 leads to some state t with $t \not\models \Phi$

DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

$\pi := \varnothing$ ⟵ stack for error indication

```
FOR ALL s0 ∈ S0 DO
      IF DFS(s0, Φ) THEN
            return "no" and reverse(π)
      FI
OD
return "yes"
```



$DFS(s_0, \Phi)$ returns "true" iff depth-first search from state s_0 leads to some state t with $t \not\models \Phi$

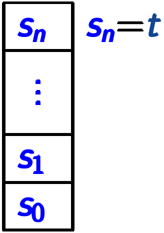
DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

$U := \varnothing$ ⟵ stores the "processed" states

$\pi := \varnothing$ ⟵ stack for error indication

```
FOR ALL s0 ∈ S0 DO
    IF DFS(s0, Φ) THEN
        return "no" and reverse(π)
    FI
OD
return "yes"
```



$DFS(s_0, \Phi)$ returns "true" iff depth-first search from state s_0 leads to some state t with $t \not\models \Phi$

Recursive algorithm $DFS(s, \Phi)$

IS2.5-8

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

```
IF s ∉ U THEN
     IF s ⊭ Φ THEN return "true" FI
     IF s ⊨ Φ THEN

          ⋮

     FI
FI
return "false"
```

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

```
IF s ∉ U THEN
    IF s ⊭ Φ THEN return "true" FI
    IF s ⊨ Φ THEN
        insert s in U;




    FI
FI
return "false"
```

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

```
IF s ∉ U THEN
    IF s ⊭ Φ THEN return "true" FI
    IF s ⊨ Φ THEN
        insert s in U;
        FOR ALL s' ∈ Post(s) DO
            IF DFS(s', Φ) THEN
                return "true" FI
        OD
    FI
FI
return "false"
```

Recursive algorithm $DFS(s, \Phi)$

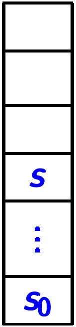
"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

```
Push(π, s);
IF s ∉ U THEN
    IF s ⊭ Φ THEN return "true" FI
    IF s ⊨ Φ THEN
        insert s in U;
        FOR ALL s' ∈ Post(s) DO
            IF DFS(s', Φ) THEN
                return "true" FI
        OD
    FI
FI
Pop(π); return "false"
```

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

```
Push(π, s);
IF s ∉ U THEN
     IF s ⊭ Φ THEN return "true" FI
     IF s ⊨ Φ THEN
          insert s in U;
          FOR ALL s' ∈ Post(s) DO
               IF DFS(s', Φ) THEN
                    return "true" FI
          OD
     FI
FI
Pop(π); return "false"
```

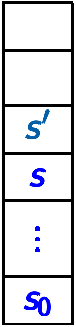


initial state

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

$Push(\pi, s)$;
IF $s \notin U$ THEN
    IF $s \not\models \Phi$ THEN return "true" FI
    IF $s \models \Phi$ THEN
        insert s in U;
        FOR ALL $s' \in Post(s)$ DO
            IF $DFS(s', \Phi)$ THEN
                return "true" FI
        OD
    FI
FI
$Pop(\pi)$; return "false"

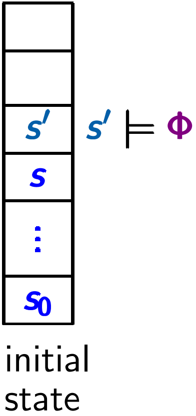


initial state

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

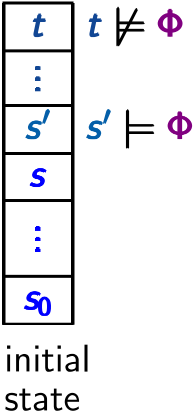
$Push(\pi, s)$;
IF $s \notin U$ THEN
  IF $s \not\models \Phi$ THEN return "true" FI
  IF $s \models \Phi$ THEN
    insert s in U;
    FOR ALL $s' \in Post(s)$ DO
      IF $DFS(s', \Phi)$ THEN
        return "true" FI
    OD
  FI
FI
$Pop(\pi)$; return "false"



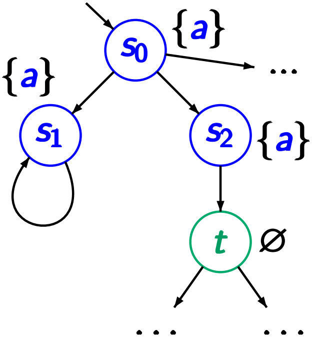
Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots s' \ldots t$ with $t \not\models \Phi$

$Push(\pi, s)$;
IF $s \notin U$ THEN
    IF $s \not\models \Phi$ THEN return "true" FI
    IF $s \models \Phi$ THEN
        insert s in U;
        FOR ALL $s' \in Post(s)$ DO
            IF $DFS(s', \Phi)$ THEN
                return "true" FI
        OD
    FI
FI
$Pop(\pi)$; return "false"



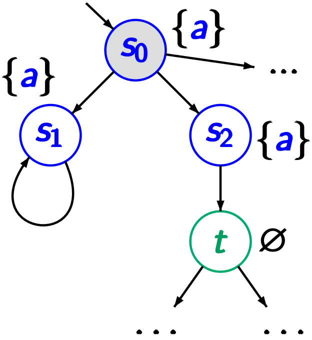
Example: invariant checking



invariant
condition a

$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

Example: invariant checking

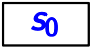


invariant condition a

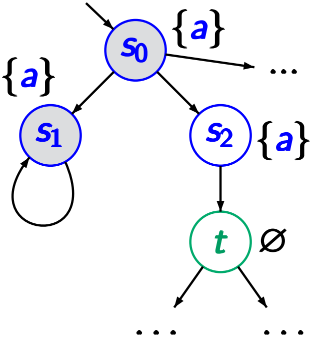
$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

$DFS(s_0, a)$

stack π



Example: invariant checking



invariant condition a

$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

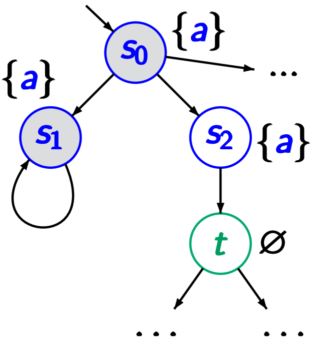
$DFS(s_0, a)$

$DFS(s_1, a)$

stack π

s_1
s_0

Example: invariant checking



invariant
condition a

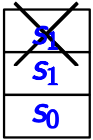
$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

$DFS(s_0, a)$

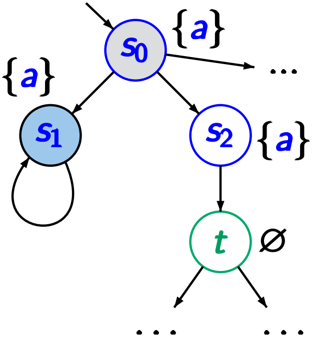
$DFS(s_1, a)$

$DFS(s_1, a)$

stack π



Example: invariant checking

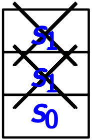


$DFS(s_0, a)$

$DFS(s_1, a)$

$DFS(s_1, a)$

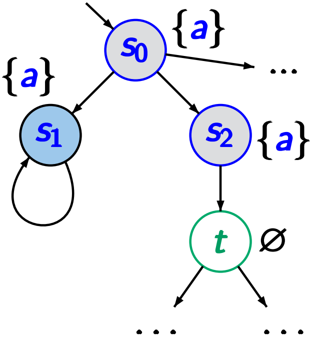
stack π



invariant condition a

$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

Example: invariant checking



invariant condition a

$s_0, s_1, s_2 \models a$
$t \not\models a$

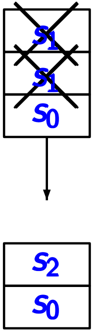
$DFS(s_0, a)$

$DFS(s_1, a)$

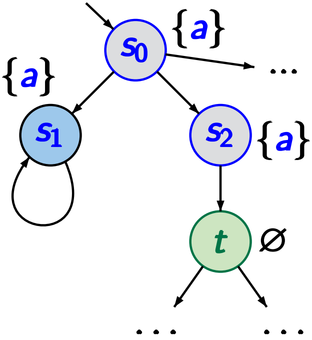
$DFS(s_1, a)$

$DFS(s_2, a)$

stack π



Example: invariant checking



invariant condition a

$s_0, s_1, s_2 \models a$

$t \not\models a$

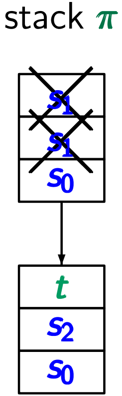
$DFS(s_0, a)$

$DFS(s_1, a)$

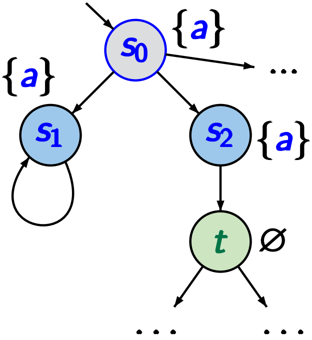
$DFS(s_1, a)$

$DFS(s_2, a)$

$DFS(t, a)$



Example: invariant checking



invariant
condition a

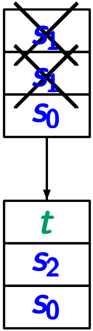
$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

$DFS(s_0, a)$

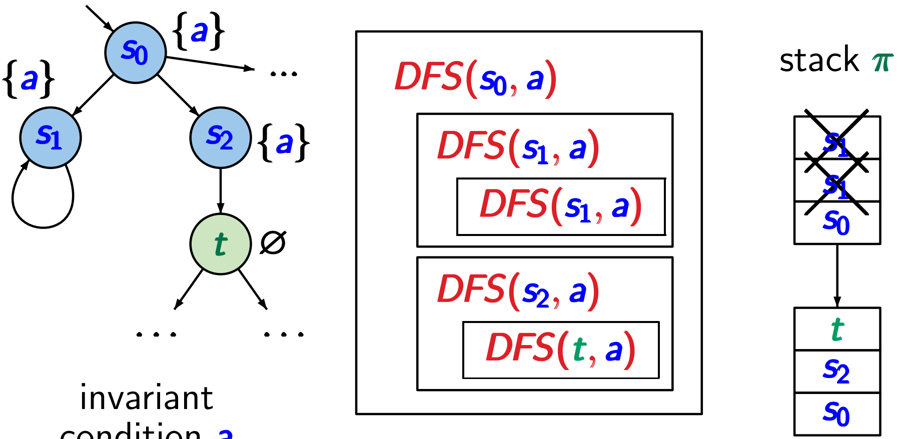
$DFS(s_1, a)$
$DFS(s_1, a)$

$DFS(s_2, a)$
$DFS(t, a)$

stack π



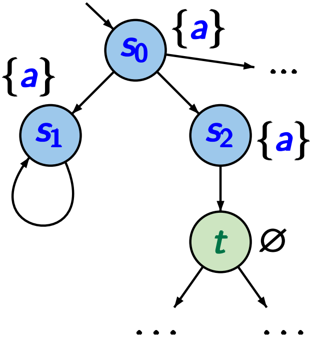
Example: invariant checking

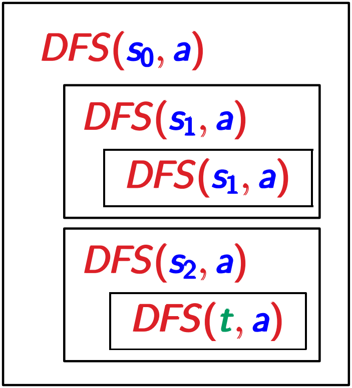


invariant condition a

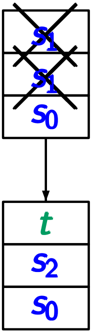
$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

Example: invariant checking





stack π



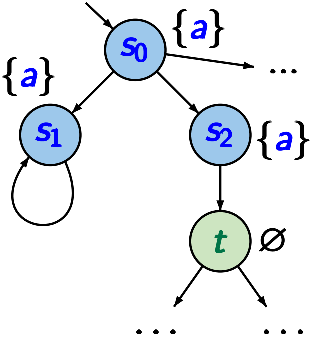
invariant condition a

$s_0, s_1, s_2 \models a$

$t \not\models a$

$s_0 \not\models$ "always a"

Example: invariant checking



invariant condition a

$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

$DFS(s_0, a)$

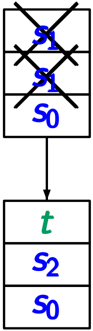
$DFS(s_1, a)$

$DFS(s_1, a)$

$DFS(s_2, a)$

$DFS(t, a)$

stack π



$s_0 \not\models$ "always a" $\longleftarrow$ error indication: $s_0\, s_2\, t$