# Model Checking Regular Safety Properties

**Lecture #8 of Model Checking**

*Joost-Pieter Katoen*

Lehrstuhl 2: Software Modeling & Verification

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

April 25, 2007

# Overview Lecture #8

⇒ Regular Safety Properties

- Finite Automata in a Nutshell
- Verifying Regular Safety Properties
  - Reduction to Invariant Checking
  - Proof of Correctness
  - The Algorithm

# Safety properties

- LT property $P_{safe}$ over *AP* is a *safety property* if
  - for all $\sigma \in \left(2^{AP}\right)^{\omega} \setminus P_{safe}$ there exists a finite prefix $\widehat{\sigma}$ of $\sigma$ such that:

$$P_{safe} \cap \left\{ \sigma' \in \left(2^{AP}\right)^{\omega} \mid \widehat{\sigma} \text{ is a prefix of } \sigma' \right\} = \varnothing$$

- The set of bad prefixes for $P_{safe}$:

$$\mathit{BadPref}(P_{safe}) = \{ \widehat{\sigma} \in \left(2^{AP}\right)^{*} \mid \forall \sigma \in \left(2^{AP}\right)^{\omega} . \widehat{\sigma}\, \sigma \notin P_{safe} \}$$

  - $P_{safe}$ is a regular safety property if $\mathit{BadPref}(P_{safe})$ is regular

Given regular safety property $P_{safe}$, how to check $TS \models P_{safe}$?

# Example regular safety properties

# Overview Lecture #8

- Regular Safety Properties

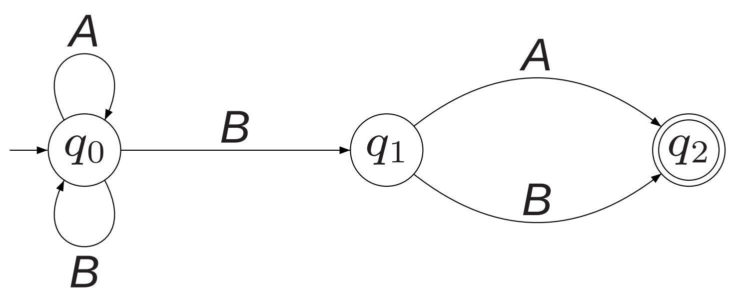
⇒ Finite Automata in a Nutshell

- Verifying Regular Safety Properties
  - Reduction to Invariant Checking
  - Proof of Correctness
  - The Algorithm

# Finite automata

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

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



# Size of an NFA

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

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

# Language of an automaton

- NFA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ and word $w = A_1 \ldots A_n \in \Sigma^*$
- A *run* for $w$ in $\mathcal{A}$ is a finite sequence $q_0 \, q_1 \, \ldots \, q_n$ such that:
  - $q_0 \in Q_0$ and $q_i \xrightarrow{A_{i+1}} q_{i+1}$ for all $0 \leqslant i < n$
- Run $q_0 \, q_1 \, \ldots \, q_n$ is *accepting* if $q_n \in F$
- $w \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if there exists an accepting run for $w$
- The *accepted language* of $\mathcal{A}$:

$$\mathcal{L}(\mathcal{A}) = \left\{ w \in \Sigma^* \mid \text{ there exists an accepting run for } w \text{ in } \mathcal{A} \right\}$$

- NFA $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$

# Example runs and accepted words

# Accepted language revisited

Extend the transition function $\delta$ to $\delta^* : Q \times \Sigma^* \to 2^Q$ by:

$$\delta^*(q, \varepsilon) = \{ q \} \quad \text{and} \quad \delta^*(q, A) = \delta(q, A)$$

$$\delta^*(q, A_1 A_2 \ldots A_n) = \bigcup_{p \in \delta(q, A_1)} \delta^*(p, A_2 \ldots A_n)$$

$\delta^*(q, w)$ = set of states reachable from $q$ for the word $w$

Then: $\mathcal{L}(\mathcal{A}) = \left\{ w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \varnothing \text{ for some } q_0 \in Q_0 \right\}$

The class of languages accepted by NFA (over $\Sigma$)

= the class of regular languages (over $\Sigma$)

# Intersection

- Let NFA $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, Q_{0,i}, F_i)$, with $i{=}1, 2$
- The *product automaton*

$$\mathcal{A}_1 \otimes \mathcal{A}_2 \;=\; (Q_1 \times Q_2, \Sigma, \delta, Q_{0,1} \times Q_{0,2}, F_1 \times F_2)$$

where $\delta$ is defined by:

$$\frac{q_1 \xrightarrow{A}_1 q_1' \;\wedge\; q_2 \xrightarrow{A}_2 q_2'}{(q_1, q_2) \xrightarrow{A} (q_1', q_2')}$$

- Well-known result: $\mathcal{L}(\mathcal{A}_1 \otimes \mathcal{A}_2) \;=\; \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$

# Total NFA

Automaton $\mathcal{A}$ is called *deterministic* if

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

DFA $\mathcal{A}$ is called *total* if

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

*any DFA can be turned into an equivalent total DFA*

*total DFA provide unique successor states, and thus, unique runs for each input word*

# Determinization

For NFA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ let $\mathcal{A}_{det} = (2^Q, \Sigma, \delta_{det}, Q_0, F_{det})$ with:

$$F_{det} = \{Q' \subseteq Q \mid Q' \cap F \neq \varnothing\}$$

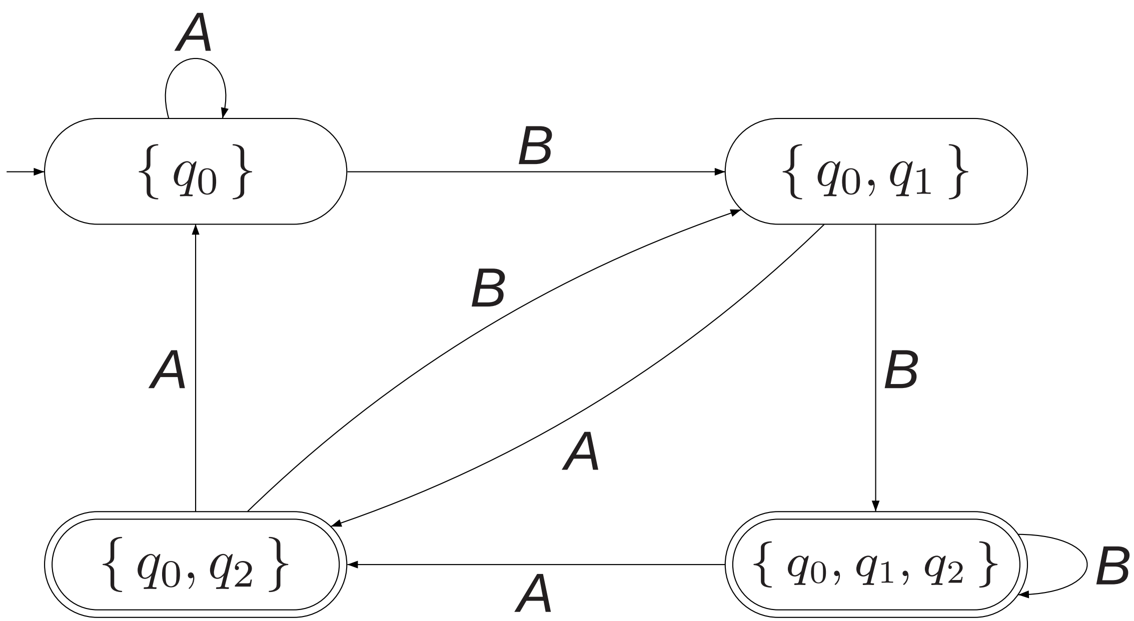
and the total transition function $\delta_{det} : 2^Q \times \Sigma \rightarrow 2^Q$ is defined by:

$$\delta_{det}(Q', A) = \bigcup_{q \in Q'} \delta(q, A)$$

$\mathcal{A}_{det}$ is a total DFA and, for all $w \in \Sigma^*$: $\delta^*_{det}(Q_0, w) = \bigcup_{q_0 \in Q_0} \delta^*(q_0, w)$

Thus: $\mathcal{L}(\mathcal{A}_{det}) = \mathcal{L}(\mathcal{A})$

# Determinization




*a deterministic finite automaton accepting* $\mathcal{L}((A + B)^* B(A + B))$

# Facts about finite automata

- They are as expressive as regular languages
- They are closed under $\cap$ and complementation
  - NFA $\mathcal{A} \otimes B$ (= cross product) accepts $\mathcal{L}(A) \cap \mathcal{L}(B)$
  - Total DFA $\overline{\mathcal{A}}$ (= swap all accept and normal states) accepts $\overline{\mathcal{L}(A)} = \Sigma^* \setminus \mathcal{L}(\mathcal{A})$
- They are closed under determinization (= removal of choice)
  - although at an exponential cost.....
- $\mathcal{L}(\mathcal{A}) = \varnothing$? = check for reachable accept state in $\mathcal{A}$
  - this can be done using a simple depth-first search
- For regular language $\mathcal{L}$ there is a unique minimal DFA accepting $\mathcal{L}$

# Overview Lecture #8

- Regular Safety Properties
- Finite Automata in a Nutshell

⇒ Verifying Regular Safety Properties
  - Reduction to Invariant Checking
  - Proof of Correctness
  - The Algorithm

# Peterson's banking system

Person Left behaves as follows:

**while** true $\{$

$\ldots\ldots$

$rq$ : $b_1, x = \text{true}, 2;$

$wt$ : $\textbf{wait until}(x == 1 \,||\, \neg\, b_2)\ \{$

$cs$ : $\ldots$ @account$_L \ldots\}$

$b_1 = \text{false};$

$\ldots\ldots$

$\}$

Person Right behaves as follows:

**while** true $\{$

$\ldots\ldots$

$rq$ : $b_2, x = \text{true}, 1;$

$wt$ : $\textbf{wait until}(x == 2 \,||\, \neg\, b_1)\ \{$

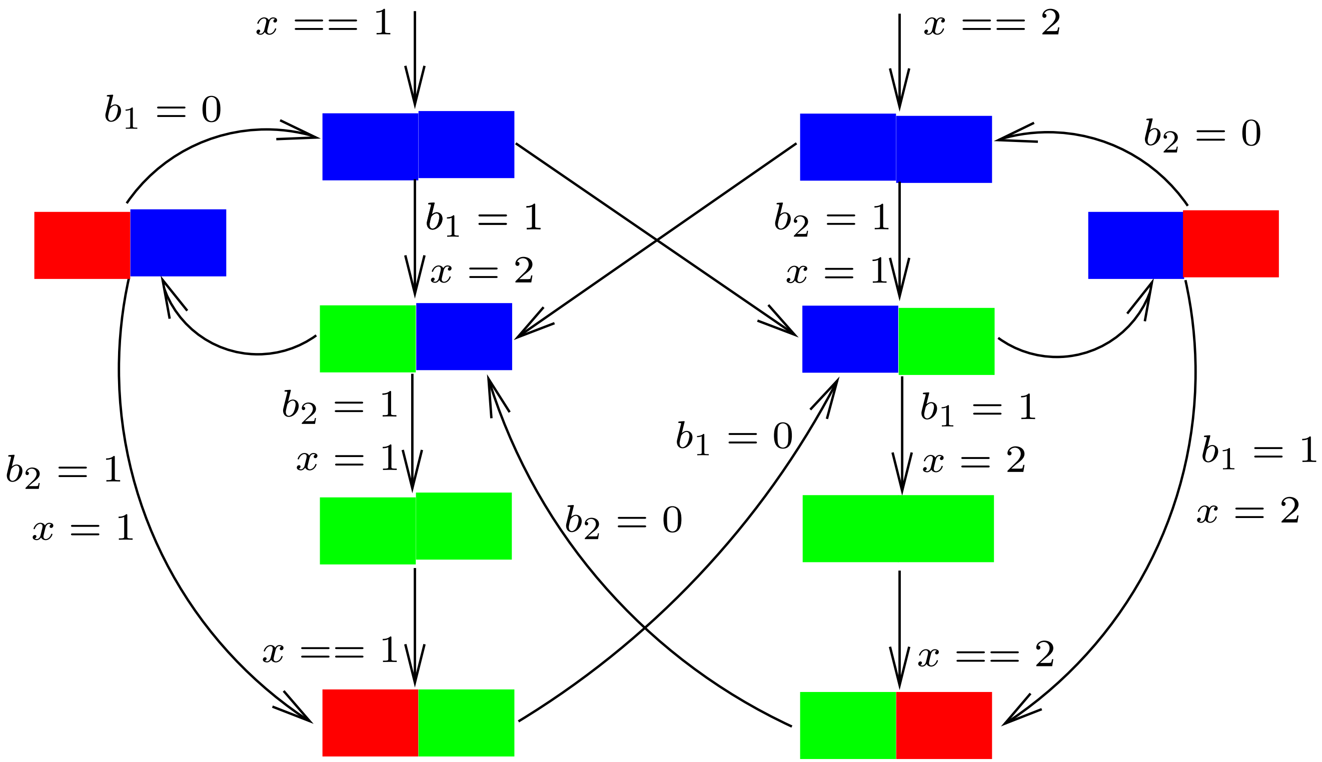
$cs$ : $\ldots$ @account$_R \ldots\}$

$b_2 = \text{false};$

$\ldots\ldots$

$\}$

# Is the banking system safe?




Can we guarantee that only one person at a time has access to the bank account?

"always $\neg$ (@account$_L$ $\wedge$ @account$_R$)"

# Is the banking system safe?

- Safe = at most one person may have access to the account
- Unsafe: two have access to the account simultaneously
  - unsafe behaviour can be characterized by bad prefix
  - alternatively (in this case) by the finite automaton:


- Checking safety: $\mathit{Traces}(\mathit{System}) \cap \mathit{BadPref}(P_{safe}) = \varnothing$?
  - intersection, complementation and emptiness of languages . . .

# Regular safety properties

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

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

*every invariant is regular*

# Problem statement

Let

- $P_{safe}$ be a regular safety property over *AP*
- $\mathcal{A}$ an NFA recognizing the bad prefixes of $P_{safe}$
  - assume that $\varepsilon \notin \mathcal{L}(\mathcal{A})$

  $\Rightarrow$ otherwise all finite words over $2^{AP}$ are bad prefixes
- *TS* a *finite* transition system (over *AP*) without terminal states

How to establish whether $TS \models P_{safe}$?

# Basic idea of the algorithm

$TS \models P_{safe}$ if and only if $Traces_{fin}(TS) \cap BadPref(P_{safe}) = \varnothing$

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

if and only if $TS \otimes \mathcal{A} \models$ "always" $\Phi$    to be proven

*But . . . . . . this amounts to invariant checking on TS* $\otimes\, \mathcal{A}$

$\Rightarrow$ *checking regular safety properties can be done by depth-first search!*

# Synchronous product (revisited)

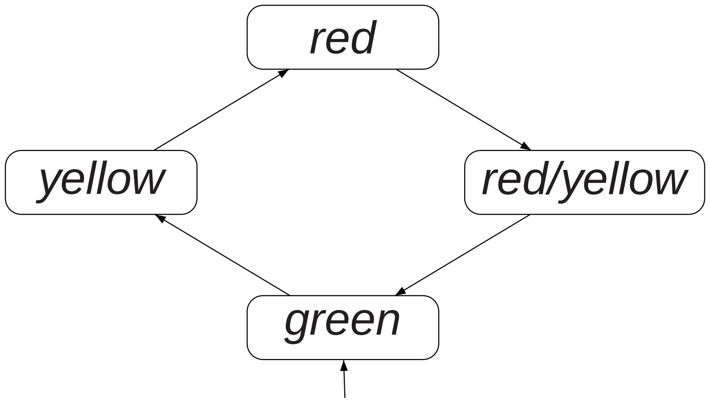
For transition system $TS = (S, Act, \rightarrow, I, AP, L)$ without terminal states and $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ an NFA with $\Sigma = 2^{AP}$ and $Q_0 \cap F = \varnothing$, let:

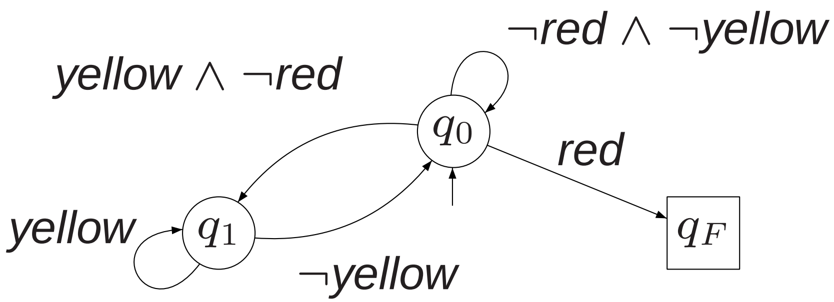
$$TS \otimes \mathcal{A} \;=\; (S', Act, \rightarrow', I', AP', L') \qquad \text{where}$$

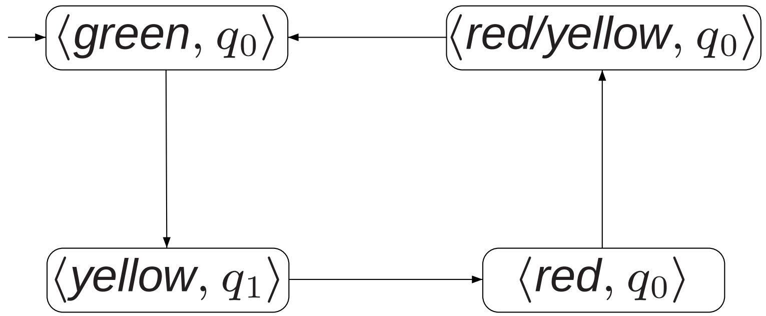
- $S' = S \times Q$, $AP' = Q$ and $L'(\langle s, q \rangle) \;=\; \{ \, q \, \}$
- $\rightarrow'$ is the smallest relation defined by: $\dfrac{s \xrightarrow{\alpha} t \;\wedge\; q \xrightarrow{L(t)} p}{\langle s, q \rangle \xrightarrow{\alpha}{}' \langle t, p \rangle}$
- $I' = \{ \, \langle s_0, q \rangle \mid s_0 \in I \;\wedge\; \exists q_0 \in Q_0. \; q_0 \xrightarrow{L(s_0)} q \, \}$

*without loss of generality it may be assumed that $TS \otimes \mathcal{A}$ has no terminal states*

# Example product



# Verification of 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* $P_{inv(A)} = \bigwedge_{q \in F} \neg q$

# Counterexamples

For each initial path fragment $\langle s_0, q_1 \rangle \ldots \langle s_n, q_{n+1} \rangle$ of *TS* $\otimes \mathcal{A}$:

$$q_1, \ldots, q_n \notin F \text{ and } q_{n+1} \in F \quad \Rightarrow \quad \underbrace{\textit{trace}(s_0\, s_1 \ldots s_n)}_{\text{bad prefix for } P_{safe}} \in \mathcal{L}(\mathcal{A})$$

# Verification algorithm

*Input:* finite transition system *TS* and regular safety property $P_{safe}$
*Output:* true if *TS* $\models P_{safe}$. Otherwise false plus a counterexample for $P_{safe}$.

---

Let NFA $\mathcal{A}$ (with accept states $F$) be such that $\mathcal{L}(\mathcal{A}) =$ *BadPref*$(P_{safe})$;
Construct the product transition system *TS* $\otimes \mathcal{A}$;
Check the invariant $P_{inv(\mathcal{A})}$ with proposition $\neg F = \bigwedge_{q \in F} \neg q$ on *TS* $\otimes \mathcal{A}$

**if** *TS* $\otimes \mathcal{A} \models P_{inv(\mathcal{A})}$ **then**
  **return** true
**else**
  Determine initial path fragment $\langle s_0, q_1 \rangle \ldots \langle s_n, q_{n+1} \rangle$ of *TS* $\otimes \mathcal{A}$ with $q_{n+1} \in F$
  **return** (false, $s_0\, s_1 \ldots s_n$)
**fi**

# Example

# Time complexity

The time and space complexity of checking a regular safety property $P_{safe}$ against transition system *TS* is in:

$$\mathcal{O}(|TS| \cdot |\mathcal{A}|)$$

where $\mathcal{A}$ is an NFA recognizing the bad prefixes of $P_{safe}$

# Can time complexity be improved?

The safety property $P_{safe}$ is regular

if and only if

the set of minimal bad prefixes for $P_{safe}$ is regular

*BadPref*$(P_{safe})$ is regular if and only if *MinBadPref*$(P_{safe})$ is regular

$\Rightarrow$ use automaton for minimal bad prefixes in product construction