

# Model Checking

René Thiemann

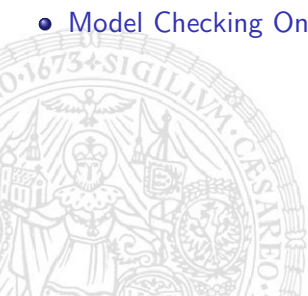
Institute of Computer Science  
University of Innsbruck

SS 2009



# Outline

- Organization & Overview
- Model Checking On-the-Fly



# Outline

- Organization & Overview
- Model Checking On-the-Fly



## Organization

- Last lecture = 1st exam
  - Some of the lectures are used solely to discuss exercises
  - Option: some weeks with 4 hours MC to finish early in semester
- ⇒ Date of exam is before “exam week”

## Literature

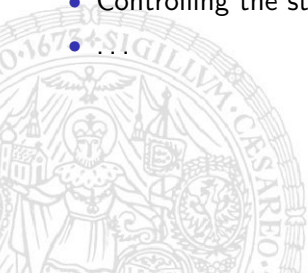
- Christel Baier and Joost-Pieter Katoen,  
**Principles of Model Checking**, MIT Press, 2008
- Edmund M. Clarke, Orna Grumberg, and Doron A. Peled,  
**Model Checking**, MIT Press, 1999
- ...

## Prerequisites

- Basic knowledge of Logic
- Basic knowledge of CTL & LTL
- Basic knowledge of Transition Systems
- Basic knowledge of Büchi Automata

# Selection of Topics

- Model checking on-the-fly (today)
- S1S
- $\mu$ -calculus
- Model checking of real-time systems
- Controlling the state-space explosion problem
- ...



# $\mu$ -Calculus

In CTL: semantics based on least and greatest fixpoint

In  $\mu$ -calculus:

- explicit least- and greatest fixpoint operators
- easy to implement
- many logics can be translated into  $\mu$ -calculus
- parallel model checking algorithms available

⇒  $\mu$ -calculus as efficient basis for model-checking for several logics

# S1S

Consider the following property:

Between every green and red phase there is at least one orange phase.

Formulating these kinds of properties in LTL is doable, but not intuitive

$$G(\text{red} \Rightarrow X(G \neg \text{green}) \vee (\neg \text{green} \wedge (X \neg \text{green} U \text{orange}))))$$

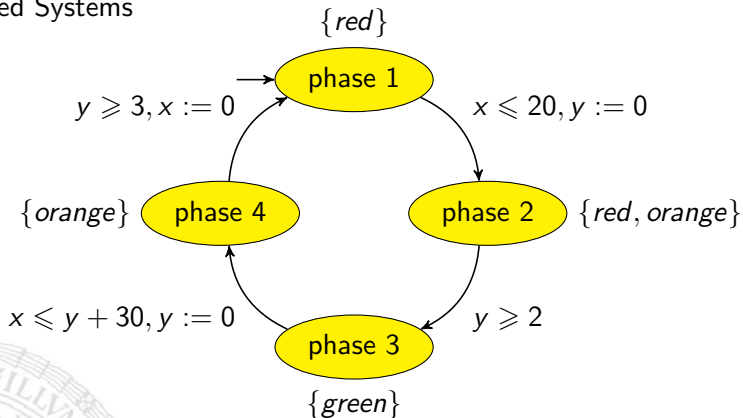
Use S1S instead:

$$\forall t_1, t_2 : (t_1 < t_2 \wedge \text{green}(t_1) \wedge \text{red}(t_2)) \Rightarrow \exists t_3 : t_1 < t_3 < t_2 \wedge \text{orange}(t_3)$$

- Allows readable and succinct specifications
- One can perform model checking using Büchi automata

# Model Checking of Real-Time Systems

- Timed Systems



- Timed Specifications

One does not have to wait more than 50 seconds for green:

$$\Phi = GF^{\leq 50} \text{green}$$



# Controlling the State-Space Explosion Problem

Reduce search space in various ways

- Abstraction:  
instead of 16-bit integer, only distinguish between even and odd, or between positive, 0, negative, or between . . .
- Partial order reduction:  
if process 1 and process 2 perform operations on **local** variables, then schedule process 1 always before process 2  
⇒ less interleaving, smaller transition system

• . . .

# Outline

- Organization & Overview
- Model Checking On-the-Fly



# LTL Model Checking

Given: Transition system  $TS$ , LTL-formula  $\varphi$

- $TS \models \varphi$  iff  $\mathcal{L}(TS) \subseteq \mathcal{L}(\varphi)$
- Algorithmic Solution (IMC):  
 Build **N**on-deterministic **B**üchi **A**utomata  $\mathcal{A}_{\neg\varphi}$  with  $\mathcal{L}(\mathcal{A}_{\neg\varphi}) = \overline{\mathcal{L}(\varphi)}$   
 Build intersection NBA:  $\mathcal{B} = TS \otimes \mathcal{A}_{\neg\varphi}$

$$\mathcal{L}(\mathcal{B}) = \mathcal{L}(TS) \setminus \mathcal{L}(\varphi)$$

Finally check  $\mathcal{L}(\mathcal{B}) = \emptyset$



# Non-deterministic Büchi Automata

- Remember: NBA  $\mathcal{A}$  is 5-tuple  $(Q, \Sigma, q_0, \delta, F)$ 
  - $Q$ : finite set of states
  - $\Sigma$ : finite set of letters, input alphabet
  - $q_0 \in Q$ : initial state
  - $\delta : Q \times \Sigma \rightarrow 2^Q$ : transition function
  - $F \subseteq Q$ : final (accepting) states
- Run** for  $w = a_0 a_1 a_2 \dots \in \Sigma^\omega$  is infinite sequence  $q_0 q_1 q_2 \dots$  with

$$q_{i+1} \in \delta(q_i, a_i) \quad (q_i \xrightarrow{a_i} q_{i+1}) \quad \text{for all } i \in \mathbb{N}$$

- Run  $q_0 q_1 \dots q_n \dots$  is **accepting** if for infinitely many  $i$ :  $q_i \in F$
- $w \in \Sigma^\omega$  is **accepted** by  $\mathcal{A}$  if there exists an accepting run for  $w$
- The **accepted language** of  $\mathcal{A}$ :

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

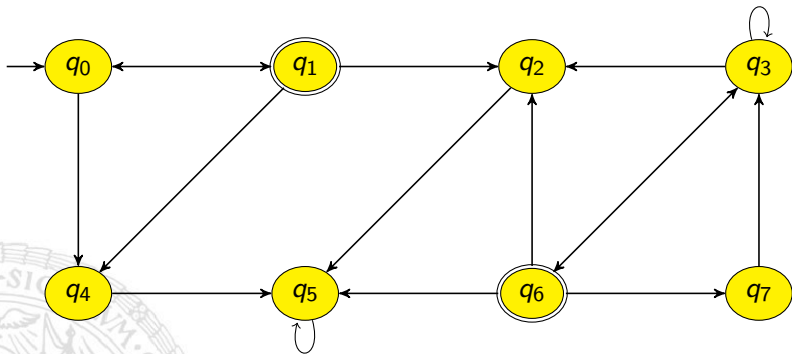
## Checking Emptiness of NBAs

- For checking emptiness, input letters can be ignored
- ⇒ Obtain finite graph from NBA
- Since  $Q$  is finite, each infinite run must end in **cycle**
- $\mathcal{C} \subseteq Q$  is **cycle** iff every state of  $\mathcal{C}$  is reachable from every state of  $\mathcal{C}$
- ⇒  $\mathcal{L}(\mathcal{A}) \neq \emptyset$  iff  
 $\mathcal{A}$  has path from initial state to cycle  $\mathcal{C}$  which contains final state

### Solution via Strongly Connected Components

1. Compute **SCCs** (maximal cycles) of  $\mathcal{A}$  by Tarjan's algorithm
  2. Perform **depth first search** (DFS) to determine **reachable SCCs**
  3.  $\mathcal{L}(\mathcal{A}) \neq \emptyset$  iff one of the reachable SCCs contains final state
- ⇒ Linear time complexity (optimal)
- ⇒ Complete graph is required in step 1

# Example



# Checking Emptiness of NBAs

## Solution via Strongly Connected Components

1. Compute **SCCs** (maximal cycles) of  $\mathcal{A}$  by Tarjan's algorithm
  2. Perform DFS to determine **reachable SCCs**
  3.  $\mathcal{L}(\mathcal{A}) \neq \emptyset$  iff one of the reachable SCCs contains final state
- ⇒ Linear time complexity (optimal)
- ⇒ Complete NBA is required for step 1

## Naive On-the-Fly Solution

1. Compute reachable final states  $R_F$  by **outer DFS**
  2. For each visited  $q \in R_F$  in step 1 **directly** check whether it belongs to a cycle by an **inner DFS**
- ⇒ Complete NBA not required
- ⇒ only parts of NBA have to be generated during DFSs (on-the-fly)

# Naive Algorithm

outer\_dfs( $q_0$ )

**terminate**(true) // Yes,  $\mathcal{L}(\mathcal{A}) = \emptyset$

**procedure** outer\_dfs( $q$ )

mark( $q$ )

**if**  $q \in F$  **then** inner\_dfs( $q$ )

**for all** successors  $q'$  of  $q$  **do**

**if**  $q'$  not marked **then** outer\_dfs( $q'$ )

**procedure** inner\_dfs( $q$ )

flag( $q$ ) // for each outside call of inner\_dfs( $q$ ) new flags are used

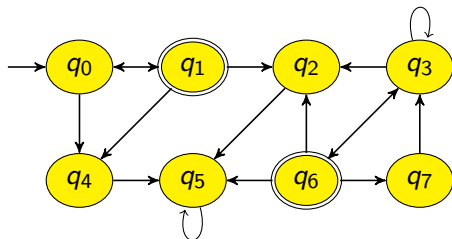
**for all** successors  $q'$  of  $q$  **do**

**if**  $q'$  on outer\_dfs-stack **then terminate**(false) //  $\mathcal{L}(\mathcal{A}) \neq \emptyset$

**else if**  $q'$  not flagged **then** inner\_dfs( $q'$ )



# Example



# Example ( $2n + 1$ states)



Example (but  $\geq n^2$  steps)

outer_dfs-stack	inner_dfs-stack	marked	flagged
$\varepsilon$	—	$\emptyset$	—
$q_0$	—	$\{q_0\}$	—
$f_1 q_0$	—	$\{q_0, f_1\}$	—
$f_1 q_0$	$f_1$	$\{q_0, f_1\}$	$\{f_1\}$
		...	
$f_1 q_0$	$q_n \dots q_1 f_1$	$\{q_0, f_1\}$	$\{f_1, q_1, \dots, q_n\}$
		...	
$f_1 q_0$	$f_1$	$\{q_0, f_1\}$	$\{f_1, q_1, \dots, q_n\}$
$f_1 q_0$	—	$\{q_0, f_1\}$	—
$q_1 f_1 q_0$	—	$\{q_0, f_1, q_1\}$	—
		...	
$q_n \dots q_1 f_1 q_0$	—	$\{q_0, f_1, q_1, \dots, q_n\}$	—
		...	
$f_2 q_0$	$f_2$	$\{q_0, f_1, f_2, q_1, \dots, q_n\}$	$\{f_2\}$

Now for every  $f_2, \dots, f_n$  one visits all states  $q_1, \dots, q_n$  again

# Linear On-the-Fly Algorithm for Emptiness of NBAs

outer\_dfs( $q_0$ )

**terminate**(true) // Yes,  $\mathcal{L}(\mathcal{A}) = \emptyset$

**procedure** outer\_dfs( $q$ )

mark( $q$ )

**if**  $q \in F$  **then** inner\_dfs( $q$ )

**for all** successors  $q'$  of  $q$  **do**

**if**  $q'$  not marked **then** outer\_dfs( $q'$ )

**procedure** inner\_dfs( $q$ )

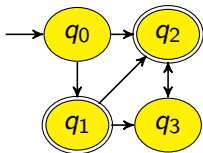
flag( $q$ ) // keep flags

**for all** successors  $q'$  of  $q$  **do**

**if**  $q'$  on outer\_dfs-stack **then** **terminate**(false) //  $\mathcal{L}(\mathcal{A}) \neq \emptyset$

**else if**  $q'$  not flagged **then** inner\_dfs( $q'$ )

## Example (Soundness of Linear On-the-Fly Algorithm)



outer_dfs-stack	inner_dfs-stack	marked	flagged
$\varepsilon$	—	$\emptyset$	$\emptyset$
$q_0$	—	$\{q_0\}$	$\emptyset$
$q_1 q_0$	—	$\{q_0, q_1\}$	$\emptyset$
$q_1 q_0$	$q_1$	$\{q_0, q_1\}$	$\{q_1\}$
$q_1 q_0$	$q_2 q_1$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
$q_1 q_0$	$q_3 q_2 q_1$	$\{q_0, q_1\}$	$\{q_1, q_2, q_3\}$
$q_2 q_1 q_0$	—	$\{q_0, q_2, q_1\}$	$\{q_1, q_2, q_3\}$
$q_2 q_1 q_0$	$q_2$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2, q_3\}$
$q_3 q_2 q_1 q_0$	—	$\{q_0, q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$

**terminate(true)**

## Correct Linear On-the-Fly Algorithm [Yannakakis et. al]

outer\_dfs( $q_0$ )

**terminate**(true) // Yes,  $\mathcal{L}(\mathcal{A}) = \emptyset$

**procedure** outer\_dfs( $q$ )

mark( $q$ )

**for all** successors  $q'$  of  $q$  **do**

**if**  $q'$  not marked **then** outer\_dfs( $q'$ )

**if**  $q \in F$  **then** inner\_dfs( $q$ )

**procedure** inner\_dfs( $q$ )

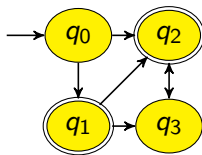
flag( $q$ ) // keep flags

**for all** successors  $q'$  of  $q$  **do**

**if**  $q'$  on outer\_dfs-stack **then** **terminate**(false) //  $\mathcal{L}(\mathcal{A}) \neq \emptyset$

**else if**  $q'$  not flagged **then** inner\_dfs( $q'$ )

# Example (Soundness of Linear On-the-Fly Algorithm)



# Soundness of the Linear On-the-Fly Algorithm

## Theorem (Yannakakis et. al)

If the result of the algorithm is false then  $\mathcal{L}(\mathcal{A}) \neq \emptyset$  and a word  $w \in \mathcal{L}(\mathcal{A})$  can be constructed. Otherwise,  $\mathcal{L}(\mathcal{A}) = \emptyset$ .

## Proof.

Easy direction:

If the algorithm terminates with false then

- outer DFS stack is  $q_n q_{n-1} \dots q_0$
- $q_n \in F$
- inner DFS stack is  $q_{m+n} q_{m-1+n} \dots q_n$
- $q_i$  is successor of  $q_{m+n}$  where  $i \leq n$

$\Rightarrow q_0 \dots q_n \dots q_{n+m} q_i \dots q_n \dots q_{n+m} q_i \dots$  is infinite and accepting run

$\Rightarrow$  Reading the letters of the corresponding transitions yields  $w$





# Model Checking On-the-Fly

- Up to now: Emptiness of NBAs on-the-fly
- Model checking  $TS \models \varphi$  is done by checking  $\mathcal{L}(\mathcal{B}) = \emptyset$  for NBA  $\mathcal{B} = TS \otimes \mathcal{A}_{\neg\varphi}$  (accepts  $\mathcal{L}(TS) \cap \mathcal{L}(\neg\varphi)$ )
- $TS = (S, \rightarrow, I, AP, L)$  is often large, but can be generated on-the-fly  
 Provided operations:
  - **init\_states()** returns set  $I \subseteq S$  of initial states
  - **succ\_states(s)** returns set of successors of  $s$  (w.r.t.  $\rightarrow$ )
  - **label\_state(s)** returns set  $L(s) \in 2^{AP}$  of atomic props. satisfied in  $s$
- $\mathcal{A}_{\neg\varphi} = (Q, 2^{AP}, q_0, \delta, F)$  is usually small and will be fully constructed
- Problem: How to generate  $\mathcal{B} = (Q', 2^{AP}, q'_0, \delta', F')$  step-by-step?  
 Required operations for emptiness-check:
  - **init\_state()** returns the initial state  $q'_0$  of  $\mathcal{B}$
  - **succ\_states(q)** returns set of successors of  $q$  (w.r.t.  $\delta'$ )
  - **final\_state(q)** returns whether  $q$  is final state of  $\mathcal{B}$

## Intersection NBA On-the-Fly

Let  $TS = (S, \rightarrow, I, AP, L)$  and  $\mathcal{A}_{\neg\varphi} = (Q, 2^{AP}, q_0, \delta, F)$ .

Then  $\mathcal{B} = TS \otimes \mathcal{A}_{\neg\varphi}$  is defined as

$$((S \times Q) \uplus \{q'_0\}, 2^{AP}, \delta', q'_0, S \times F) \quad \text{with } \delta' :$$

- $\delta'((s, q), A) = \{(s', q') \mid L(s) = A, s \rightarrow s', q' \in \delta(q, A)\}$
- $\delta'(q'_0, A) = \{(s', q') \mid s \in I, L(s) = A, s \rightarrow s', q' \in \delta(q_0, A)\}$

Thus the required operations of  $\mathcal{B}$  can be implemented as follows:

- $\text{init\_state}() = q'_0$
- $\text{final\_state}(q'_0) = \text{false}$       and       $\text{final\_state}((s, q)) = q \in F$
- $\text{succ\_states}((s, q)) = \text{succ\_states}(s) \times \underbrace{\delta(q, \text{label\_state}(s))}_{\text{Compute this first, maybe } \emptyset}$       and

$$\text{succ\_states}(q'_0) = \bigcup_{s \in \text{init\_states}()} \text{succ\_states}(s) \times \delta(q_0, \text{label\_state}(s))$$

Nice side-effect: Only **reachable part** of  $\mathcal{B}$  is created!

## Example

