Model Checking

René Thiemann

Institute of Computer Science
University of Innsbruck

SS 2008

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

$\Rightarrow$ 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
- μ-calculus
- Model checking of real-time systems
- Controlling the state-space explosion problem
- ...

μ-Calculus

In CTL: semantics based on least and greatest fixpoint

In μ-calculus:

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

$\Rightarrow$ μ-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

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

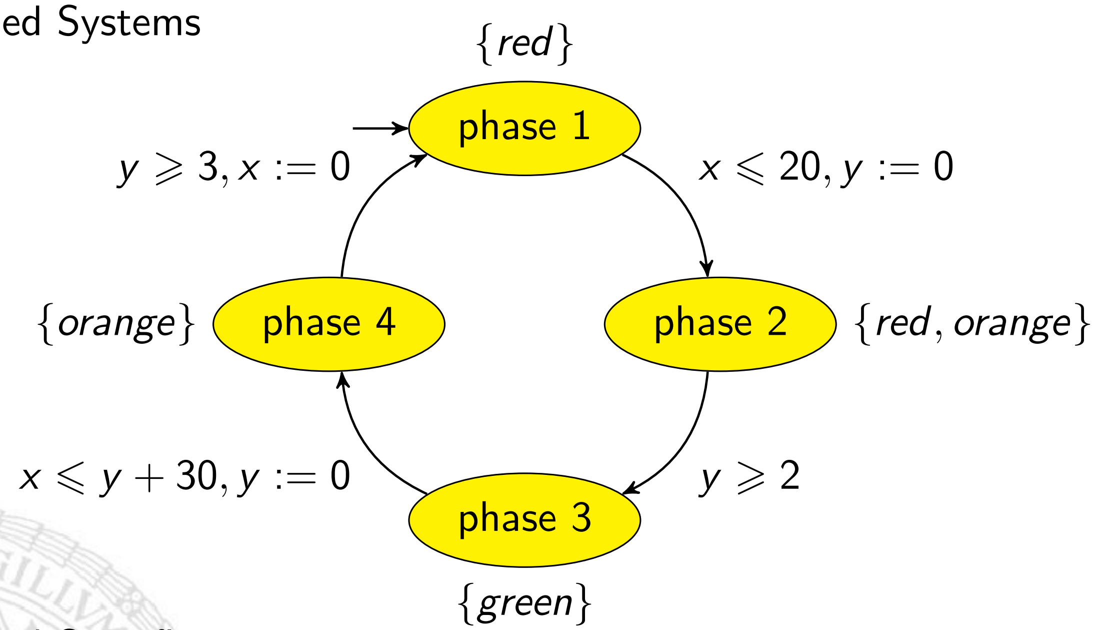
Use S1S instead:

$$\forall t_1, t_2 : (t_1 < t_2 \wedge \mathsf{green}(\mathsf{t}_1) \wedge \mathsf{red}(\mathsf{t}_2)) \Rightarrow \exists t_3 : t_1 < t_3 < t_2 \wedge \mathsf{orange}(\mathsf{t}_3)$$

- 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:

$$\Phi = \mathsf{G}\,\mathsf{F}^{\leqslant 50}\text{green}$$

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

$\Rightarrow$ less interleaving, smaller transition system

- ...

LTL Model Checking

Given: Transition system TS, LTL-formula φ

- $TS \models \varphi$ iff $\mathcal{L}(TS) \subseteq \mathcal{L}(\varphi)$
- Algorithmic Solution (IMC):
  Build Non-deterministic Büchi Automata $\mathcal{A}_{\neg\varphi}$ 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 $\mathcal{L}(\mathcal{B}) = \varnothing$

Non-deterministic Büchi Automata

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

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

- Run $q_0\, q_1\, \ldots\, q_n\, \ldots$ 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

$\Rightarrow$ Obtain finite graph from NBA

- Since $\mathcal{Q}$ is finite, each infinite run must end in cycle
- $\mathcal{C} \subseteq \mathcal{Q}$ is cycle iff every state of $\mathcal{C}$ is reachable from every state of $\mathcal{C}$

$\Rightarrow$ $\mathcal{L}(\mathcal{A}) \neq \varnothing$ iff
$\mathcal{A}$ has path from initial state to cycle $\mathcal{C}$ which contains final state

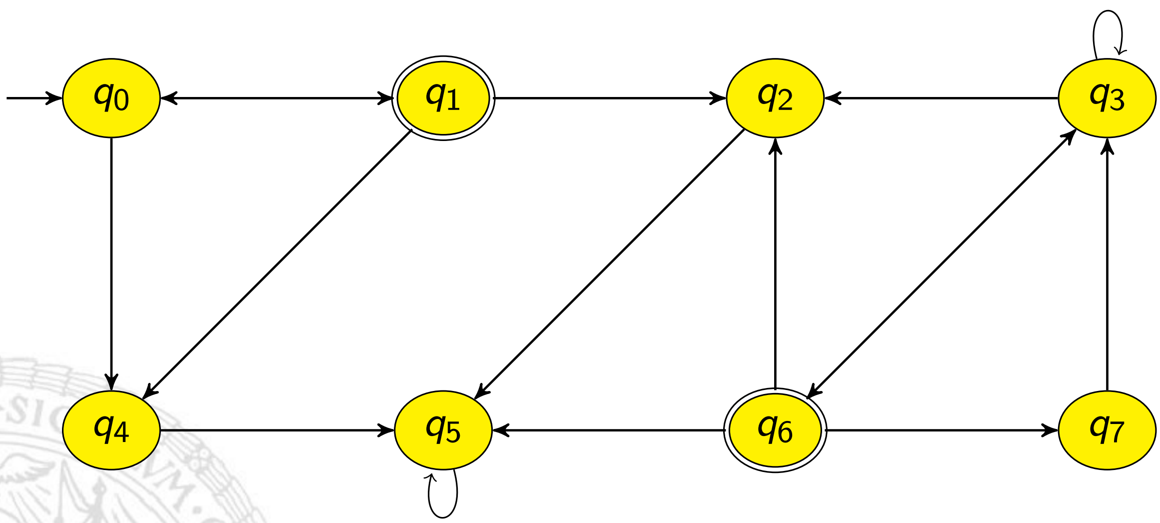
Solution via Strongly Connected Components

1. Compute SCCs (maximal cycles) of $\mathcal{A}$ by Tarjan's algorithm
2. Perform depth first search (DFS) to determine reachable SCCs
3. $\mathcal{L}(\mathcal{A}) \neq \varnothing$ iff one of the reachable SCCs contains final state

$\Rightarrow$ Linear time complexity (optimal)

$\Rightarrow$ 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. $\mathcal{L}(\mathcal{A}) \neq \varnothing$ iff one of the reachable SCCs contains final state

$\Rightarrow$ Linear time complexity (optimal)

$\Rightarrow$ 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

$\Rightarrow$ Complete NBA not required

$\Rightarrow$ only parts of NBA have to be generated during DFSs (on-the-fly)

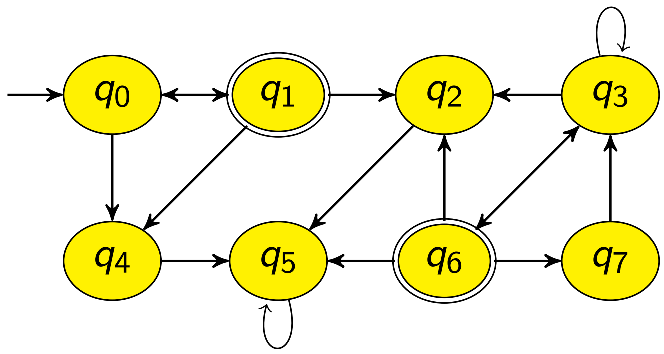
Naive Algorithm

outer_dfs(q_0)
terminate(true) // Yes, $\mathcal{L}(\mathcal{A}) = \varnothing$

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) // $\mathcal{L}(\mathcal{A}) \neq \varnothing$
        else if q' not flagged **then** inner_dfs(q')

Example



Example ($2n + 1$ states)

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

outer_dfs-stack	inner_dfs-stack	marked	flagged
ε	–	$\varnothing$	–
q_0	–	$\{q_0\}$	–
$f_1\ q_0$	–	$\{q_0, f_1\}$	–
$f_1\ q_0$	f_1	$\{q_0, f_1\}$	$\{f_1\}$
		$\cdots$	
$f_1\ q_0$	$q_n \ldots q_1\ f_1$	$\{q_0, f_1\}$	$\{f_1, q_1, \ldots, q_n\}$
		$\cdots$	
$f_1\ q_0$	f_1	$\{q_0, f_1\}$	$\{f_1, q_1, \ldots, q_n\}$
$f_1\ q_0$	–	$\{q_0, f_1\}$	–
$q_1\ f_1\ q_0$	–	$\{q_0, f_1, q_1\}$	–
		$\cdots$	
$q_n \ldots q_1\ f_1\ q_0$	–	$\{q_0, f_1, q_1, \ldots, q_n\}$	–
		$\cdots$	
$f_2\ q_0$	f_2	$\{q_0, f_1, f_2, q_1, \ldots, q_n\}$	$\{f_2\}$

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

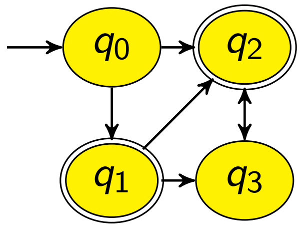
Linear On-the-Fly Algorithm for Emptyness of NBAs

outer_dfs(q_0)
terminate(true) // Yes, $\mathcal{L}(\mathcal{A}) = \varnothing$

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) // $\mathcal{L}(\mathcal{A}) \neq \varnothing$
    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
ε	$-$	$\varnothing$	$\varnothing$
q_0	$-$	$\{q_0\}$	$\varnothing$
$q_1\ q_0$	$-$	$\{q_0, q_1\}$	$\varnothing$
$q_1\ q_0$	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)

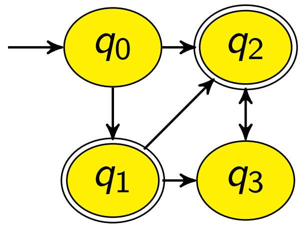
Correct Linear On-the-Fly Algorithm [Yannakakis et. al]

outer_dfs(q_0)
terminate(true) // Yes, $\mathcal{L}(\mathcal{A}) = \varnothing$

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) // $\mathcal{L}(\mathcal{A}) \neq \varnothing$
        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 $\mathcal{L}(\mathcal{A}) \neq \varnothing$ and a word $w \in \mathcal{L}(\mathcal{A})$ can be constructed. Otherwise, $\mathcal{L}(\mathcal{A}) = \varnothing$.

Proof.

Easy direction:
If the algorithm terminates with false then

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

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

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

Model Checking On-the-Fly

- Up to now: Emptyness of NBAs on-the-fly
- Model checking $TS \models \varphi$ is done by checking $\mathcal{L}(\mathcal{B}) = \varnothing$ 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
- $\mathcal{A}_{\neg\varphi} = (\mathcal{Q}, 2^{AP}, q_0, \delta, F)$ is usually small and will be fully constructed
- Problem: How to generate $\mathcal{B} = (\mathcal{Q}', 2^{AP}, q'_0, \delta', F')$ step-by-step?
  Required operations for emptyness-check:
  - init_state() returns the initial state q'_0 of $\mathcal{B}$
  - succ_states(q) returns set of successors of q (w.r.t. δ')
  - final_state(q) returns whether q is final state of $\mathcal{B}$

Intersection NBA On-the-Fly

Let $TS = (S, \to, I, AP, L)$ and $\mathcal{A}_{\neg\varphi} = (\mathcal{Q}, 2^{AP}, q_0, \delta, F)$.
Then $\mathcal{B} = TS \otimes \mathcal{A}_{\neg\varphi}$ is defined as

$$((S \times \mathcal{Q}) \uplus \{q_0'\}, 2^{AP}, \delta', q_0', S \times F) \qquad \text{with } \delta' :$$

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

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 $\text{final\_state}((s,q)) = q \in F$
- $\text{succ\_states}((s,q)) = \text{succ\_states}(s) \times \underbrace{\delta(q, \text{label\_state}(s))}_{\text{Compute this first, maybe } \varnothing}$ 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