Memory Reduction for Strategies in Infinite Games

Michael Holtmann and Christof Löding

Chair of Computer Science 7, RWTH Aachen, Germany

CIAA '07
Prague - Czech Republic
July 18, 2007

INTRODUCTION

- Infinite games are used for synthesis and verification of reactive systems
- Reactive systems
  - protocols, controllers,. . .
  - several agents with opposing objectives
  - nonterminating behavior
- Infinite games
  - the system is represented by a finite graph
  - two players (system and environment)
  - the requirements are modeled by a winning condition for either player

Introduction

- Infinite games are used for synthesis and verification of reactive systems
- Reactive systems
  - protocols, controllers,...
  - several agents with opposing objectives
  - nonterminating behavior
- Infinite games
  - the system is represented by a finite graph
  - two players (system and environment)
  - the requirements are modeled by a winning condition for either player
- Winning strategies in infinite games correspond to controller programs for reactive systems
- Two important questions:
  - What are the computational costs for solving a game?
  - What is the size of the solution (strategy automaton)?

Outline

1. Infinite Games
2. Memory Reduction
   - Minimization of Strategy Automata
   - Reduction of Game Graphs
3. Some Results

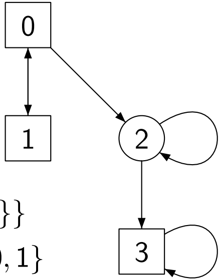
Outline

1. Infinite Games
2. Memory Reduction
   - Minimization of Strategy Automata
   - Reduction of Game Graphs
3. Some Results

Infinite Games

- Two Players: $\bigcirc$ and $\square$
- Game Graph: $G = (Q, Q_{\bigcirc}, Q_{\square}, E)$ finite and directed
- Play: Infinite path ρ through G
- Winning Condition for Player $\bigcirc$: $\varphi \subseteq Q^{\omega}$

Example: Staiger-Wagner game
Player $\bigcirc$ wins $\rho :\Longleftrightarrow \; Occ(\rho) \in \{F_1, \dots, F_k\}$
$Occ(\rho)$: set of vertices visited in ρ at least once
$\mathcal{F} = \{F_1, \dots, F_k\}$: "winning sets" for Player $\bigcirc$



$$\mathcal{F} = \{\{0,1\}, \{0,2\}, \{0,1,2,3\}\}$$

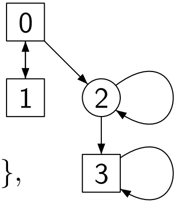
Player $\bigcirc$ wins from $W_{\bigcirc} = \{0,1\}$

Strategies and Strategy Automata

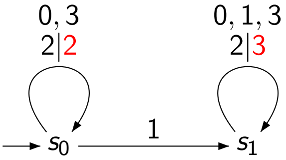
- A strategy for Player $\bigcirc$ is a function $f : Q^* Q_\bigcirc \to Q$ respecting the edge relation
- Strategy automaton: Implementation of a strategy as a finite automaton with output: $\mathcal{A} = (S, Q, s_0, \sigma, \tau)$
  - $\sigma : S \times Q \to S$ yields the memory update rule
  - $\tau : S \times Q_\bigcirc \to Q$ computes the strategy f iteratively
- Positional strategy: can be implemented by a strategy automaton with only one state (and is specified by a set $E_{pos} \subseteq E$)

Strategies and Strategy Automata

- A strategy for Player $\bigcirc$ is a function $f : Q^* Q_{\bigcirc} \to Q$ respecting the edge relation
- Strategy automaton: Implementation of a strategy as a finite automaton with output: $\mathcal{A} = (S, Q, s_0, \sigma, \tau)$
  - $\sigma : S \times Q \to S$ yields the memory update rule
  - $\tau : S \times Q_{\bigcirc} \to Q$ computes the strategy f iteratively
- Positional strategy: can be implemented by a strategy automaton with only one state (and is specified by a set $E_{pos} \subseteq E$)

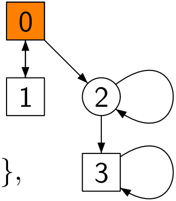


$\mathcal{F} = \{\{0,1\}, \{0,2\}, \{0,1,2,3\}\}$

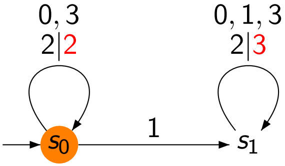


Strategies and Strategy Automata

- A strategy for Player $\bigcirc$ is a function $f : Q^* Q_{\bigcirc} \to Q$ respecting the edge relation
- Strategy automaton: Implementation of a strategy as a finite automaton with output: $\mathcal{A} = (S, Q, s_0, \sigma, \tau)$
  - $\sigma : S \times Q \to S$ yields the memory update rule
  - $\tau : S \times Q_{\bigcirc} \to Q$ computes the strategy f iteratively
- Positional strategy: can be implemented by a strategy automaton with only one state (and is specified by a set $E_{pos} \subseteq E$)



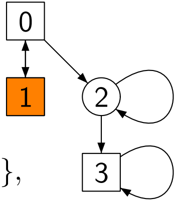
$\mathcal{F} = \{\{0,1\},\{0,2\},\{0,1,2,3\}\}$



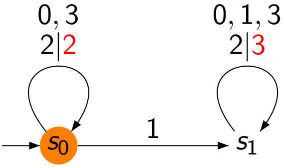
$\rho = 0$

Strategies and Strategy Automata

- A strategy for Player $\bigcirc$ is a function $f : Q^* Q_\bigcirc \to Q$ respecting the edge relation
- Strategy automaton: Implementation of a strategy as a finite automaton with output: $\mathcal{A} = (S, Q, s_0, \sigma, \tau)$
  - $\sigma : S \times Q \to S$ yields the memory update rule
  - $\tau : S \times Q_\bigcirc \to Q$ computes the strategy f iteratively
- Positional strategy: can be implemented by a strategy automaton with only one state (and is specified by a set $E_{pos} \subseteq E$)



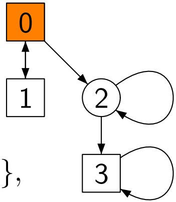
$\mathcal{F} = \{\{0, 1\}, \{0, 2\}, \{0, 1, 2, 3\}\}$



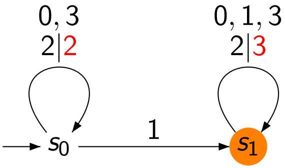
$\rho = 01$

Strategies and Strategy Automata

- A strategy for Player $\bigcirc$ is a function $f : Q^* Q_{\bigcirc} \to Q$ respecting the edge relation
- Strategy automaton: Implementation of a strategy as a finite automaton with output: $\mathcal{A} = (S, Q, s_0, \sigma, \tau)$
  - $\sigma : S \times Q \to S$ yields the memory update rule
  - $\tau : S \times Q_{\bigcirc} \to Q$ computes the strategy f iteratively
- Positional strategy: can be implemented by a strategy automaton with only one state (and is specified by a set $E_{pos} \subseteq E$)



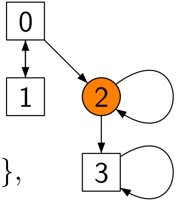
$$\mathcal{F} = \{\{0,1\},\{0,2\},\{0,1,2,3\}\}$$



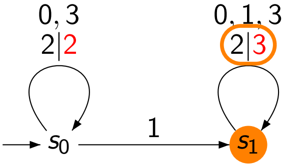
$$\rho = 010$$

Strategies and Strategy Automata

- A strategy for Player $\bigcirc$ is a function $f : Q^* Q_\bigcirc \to Q$ respecting the edge relation
- Strategy automaton: Implementation of a strategy as a finite automaton with output: $\mathcal{A} = (S, Q, s_0, \sigma, \tau)$
  - $\sigma : S \times Q \to S$ yields the memory update rule
  - $\tau : S \times Q_\bigcirc \to Q$ computes the strategy f iteratively
- Positional strategy: can be implemented by a strategy automaton with only one state (and is specified by a set $E_{pos} \subseteq E$)



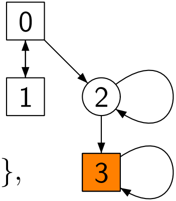
$\mathcal{F} = \{\{0, 1\}, \{0, 2\}, \{0, 1, 2, 3\}\}$



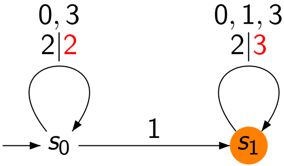
$\rho = 0102$

Strategies and Strategy Automata

- A strategy for Player $\bigcirc$ is a function $f : Q^* Q_\bigcirc \to Q$ respecting the edge relation
- Strategy automaton: Implementation of a strategy as a finite automaton with output: $\mathcal{A} = (S, Q, s_0, \sigma, \tau)$
  - $\sigma : S \times Q \to S$ yields the memory update rule
  - $\tau : S \times Q_\bigcirc \to Q$ computes the strategy f iteratively
- Positional strategy: can be implemented by a strategy automaton with only one state (and is specified by a set $E_{pos} \subseteq E$)



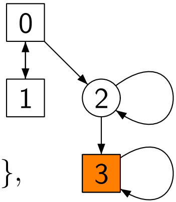
$\mathcal{F} = \{\{0,1\}, \{0,2\}, \{0,1,2,3\}\}$



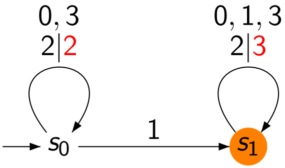
$\rho = 01023$

Strategies and Strategy Automata

- A strategy for Player $\bigcirc$ is a function $f : Q^* Q_{\bigcirc} \rightarrow Q$ respecting the edge relation
- Strategy automaton: Implementation of a strategy as a finite automaton with output: $\mathcal{A} = (S, Q, s_0, \sigma, \tau)$
  - $\sigma : S \times Q \rightarrow S$ yields the memory update rule
  - $\tau : S \times Q_{\bigcirc} \rightarrow Q$ computes the strategy f iteratively
- Positional strategy: can be implemented by a strategy automaton with only one state (and is specified by a set $E_{pos} \subseteq E$)



$$\mathcal{F} = \{\{0,1\}, \{0,2\}, \{0,1,2,3\}\}$$



$$\rho = 0102333\ldots$$

How much memory is needed?

Given: Infinite Game $\Gamma = (G, \varphi)$
Problem: Compute a winning strategy with "small" memory

Büchi,Landweber'69:
For regular winning conditions we need only finite memory

We compare two approaches to memory reduction:

- Compute strategy and then reduce corresponding automaton
  Problem: The strategy might be very complicated
- Reduce memory before strategy is computed
  Problem: How to reduce the memory?

Outline

MINIMIZATION OF STRATEGY AUTOMATA

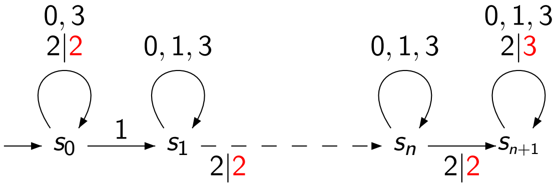
- Note: Strategy automata are Mealy machines
- Merge states from which the same output functions are computed
- Advantages:
  - Efficient
  - Independent of game graph and winning condition
- Disadvantage: The result depends on the strategy

Minimization of Strategy Automata

- Note: Strategy automata are Mealy machines
- Merge states from which the same output functions are computed
- Advantages:
  - Efficient
  - Independent of game graph and winning condition
- Disadvantage: The result depends on the strategy

Complicated strategy: Delay the move to vertex 3 for n times

$\mathcal{A}_n$:



- $\mathcal{A}_n$ counts the number of revisits to vertex 2

Game Reduction

- Idea: Simulate the given game Γ by a new game Γ' and use a solution to Γ' for solving Γ
- Extend game graph G by a (finite) memory component S
  - Often the new game graph G' is exponentially large in the size of G
  - The game Γ' admits easier winning strategies, e.g. positional ones

$$\begin{array}{c} \Gamma = (G, \varphi) \\ G = (Q, E) \end{array} \xrightarrow{\text{Game Reduction}} \begin{array}{c} \Gamma' = (G', \varphi') \\ G' = (S \times Q, E') \end{array}$$

Game Reduction

- Idea: Simulate the given game Γ by a new game Γ' and use a solution to Γ' for solving Γ
- Extend game graph G by a (finite) memory component S
  - Often the new game graph G' is exponentially large in the size of G
  - The game Γ' admits easier winning strategies, e.g. positional ones

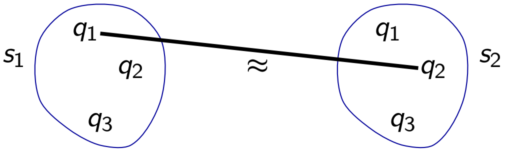
$$\begin{array}{c} \Gamma = (G, \varphi) \\ G = (Q, E) \end{array} \xrightarrow{\text{Game Reduction}} \begin{array}{c} \Gamma' = (G', \varphi') \\ G' = (S \times Q, E') \end{array}$$

Proposition: From a positional winning strategy $E'_{pos} \subseteq E'$ in Γ' we can construct a strategy automaton which implements a winning strategy in Γ

- The strategy automaton has state set S
- E' captures the memory update rule
  - $((s_1, q_1), (s_2, q_2)) \in E' \Longrightarrow \sigma(s_1, q_1) := s_2$
- The positional strategy E'_{pos} determines the output function
  - $((s_1, q_1), (s_2, q_2)) \in E'_{pos} \Longrightarrow \tau(s_1, q_1) := q_2$

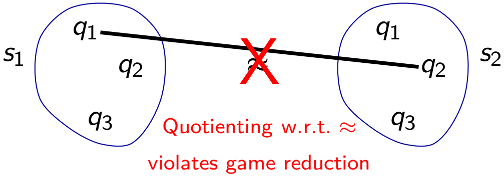
Equivalence of Memory Contents

- Note: $S \times Q$ consists of finitely many copies of Q
- Merge copies of G' s.th. properties of game reduction are preserved
- Reduce Γ' as deterministic ω-game automaton $\mathcal{A}$
  - Transition labels: $(s, q) \xrightarrow{q'} (s', q') \rightsquigarrow \mathcal{A}$ accepts the language φ
  - If $(s_1, q_1) \approx (s_2, q_2)$ for a language-preserving equivalence relation $\approx$ then from these states Player $\bigcirc$ wins exactly the same plays



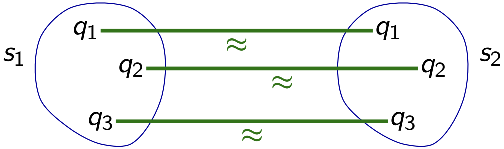
EQUIVALENCE OF MEMORY CONTENTS

- Note: $S \times Q$ consists of finitely many copies of Q
- Merge copies of G' s.th. properties of game reduction are preserved
- Reduce Γ' as deterministic ω-game automaton $\mathcal{A}$
  - Transition labels: $(s,q) \xrightarrow{q'} (s',q') \rightsquigarrow \mathcal{A}$ accepts the language φ
  - If $(s_1,q_1) \approx (s_2,q_2)$ for a language-preserving equivalence relation $\approx$ then from these states Player $\bigcirc$ wins exactly the same plays



Equivalence of Memory Contents

- Note: $S \times Q$ consists of finitely many copies of Q
- Merge copies of G' s.th. properties of game reduction are preserved
- Reduce Γ' as deterministic ω-game automaton $\mathcal{A}$
  - Transition labels: $(s,q) \xrightarrow{q'} (s',q') \rightsquigarrow \mathcal{A}$ accepts the language φ
  - If $(s_1,q_1) \approx (s_2,q_2)$ for a language-preserving equivalence relation $\approx$ then from these states Player $\bigcirc$ wins exactly the same plays



- If for all $q \in Q$ the pairs $(s_1,q),(s_2,q)$ are equivalent then s_1 and s_2 need not be distinguished

$$s_1 \approx_S s_2 :\Longleftrightarrow \forall q \in Q : (s_1,q) \approx (s_2,q)$$

- The new memory is the set $S/_{\approx_S}$

ALGORITHM

Input: $\Gamma = (G, \varphi)$ with φ regular, $G = (Q, E)$ finite

(1) Establish game reduction from $\Gamma = (G, \varphi)$ to $\Gamma' = (G', \varphi')$

(2) View Γ' as deterministic ω-automaton $\mathcal{A}$ (accepting language φ)

$$\text{Transition labels: } (s_1, q_1) \xrightarrow{q_2} (s_2, q_2)$$

(3) Reduce $\mathcal{A}$: Use equivalence relation $\approx$ on $S \times Q$ to compute $\approx_S$ on S and construct corresponding quotient automaton $\mathcal{A}/_{\approx_S}$

(4) View $\mathcal{A}/_{\approx_S}$ as infinite game Γ'' and from positional winning strategy for Player $\bigcirc$ in Γ'' compute corresponding strategy automaton for Γ

Output: Strategy Automaton for Player $\bigcirc$ from $W_\bigcirc$ in Γ

Algorithm

Input: $\Gamma = (G, \varphi)$ with φ regular, $G = (Q, E)$ finite

(1) Establish game reduction from $\Gamma = (G, \varphi)$ to $\Gamma' = (G', \varphi')$

(2) View Γ' as deterministic ω-automaton $\mathcal{A}$ (accepting language φ)

$$\text{Transition labels: } (s_1, q_1) \xrightarrow{q_2} (s_2, q_2)$$

(3) Reduce $\mathcal{A}$: Use equivalence relation $\approx$ on $S \times Q$ to compute $\approx_S$ on S and construct corresponding quotient automaton $\mathcal{A}/_{\approx_S}$

(4) View $\mathcal{A}/_{\approx_S}$ as infinite game Γ'' and from positional winning strategy for Player $\bigcirc$ in Γ'' compute corresponding strategy automaton for Γ

Output: Strategy Automaton for Player $\bigcirc$ from $W_\bigcirc$ in Γ

Theorem

Let $\Gamma = (G, \varphi)$, $\Gamma' = (G', \varphi')$ be infinite games and Γ be reducible to Γ'. If $\approx$ satisfies certain structural properties then Γ is reducible to Γ''.

Outline

Implementation

- Staiger-Wagner (= weak Muller)
  - Capture boolean combinations of safety and reachability conditions
  - Game reduction to weak Büchi games
  - A deterministic Büchi automaton is called weak if all states within the same SCC are accepting or all are rejecting
  - DWA can be minimized efficiently via minimization of DFA (Löding'01)
- Request-Response
  - $\bigwedge_{i=1}^{k}$ "If P_i is visited then now or later R_i must be visited"
  - Game reduction to Büchi games
  - Büchi automata can be reduced with delayed simulation (Etessami, Wilke,Schuller'05)
  - Also applicable to generalized Büchi and upwards-closed Muller games
- In both cases: running time exponential in the size of the given game

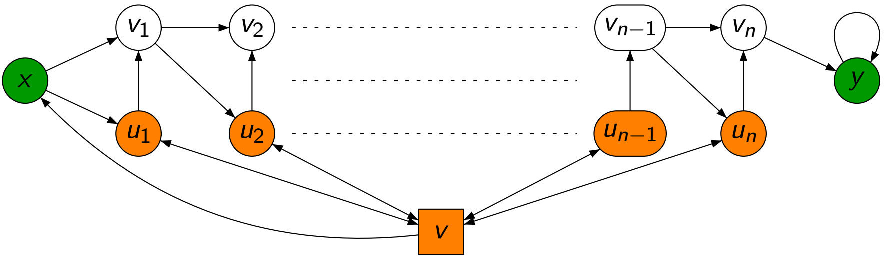
Implementation

- Staiger-Wagner (= weak Muller)
  - Capture boolean combinations of safety and reachability conditions
  - Game reduction to weak Büchi games
  - A deterministic Büchi automaton is called weak if all states within the same SCC are accepting or all are rejecting
  - DWA can be minimized efficiently via minimization of DFA (Löding'01)
- Request-Response
  - $\bigwedge_{i=1}^{k}$ "If P_i is visited then now or later R_i must be visited"
  - Game reduction to Büchi games
  - Büchi automata can be reduced with delayed simulation (Etessami, Wilke,Schuller'05)
  - Also applicable to generalized Büchi and upwards-closed Muller games
- In both cases: running time exponential in the size of the given game
- Muller, Streett
  - Game reduction to parity games
  - We use a sophisticated version of delayed simulation for which we need to solve a Büchi game (Fritz,Wilke'06)

Upper Bound

- Staiger-Wagner winning condition:

  Visit only orange vertices or both green ones



Lemma

1. *If we solve Γ'_n by a conventional algorithm (Chatterjee'06) then we get an exponential size winning strategy for Player $\bigcirc$ in Γ_n from v.*
2. *The reduced game graph computed by our Algorithm has constantly many memory contents.*

Conclusions

- Problem: How to compute winning strategies that require only a small memory?
- Classical Approach: Compute strategy and then minimize corresponding automaton
  - Reduce strategy automaton as Mealy machine
  - Advantage: efficient and independent of underlying game
  - Drawback: depends on the strategy
- Our Approach: Reduce memory and then compute strategy
  - Introduce memory (by game reduction) and compute equivalent memory contents via transformation to ω-automaton
  - Advantage: independent of winning strategies
  - Drawback: efficient minimization of ω-automata is difficult
  - Minimal ω-automaton does not guarantee optimal memory
- Experiments have shown strengths and weaknesses of both the two approaches