Good-for-Games Automata versus Deterministic Automata.

Denis Kuperberg Michał Skrzypczak

TUM Munich and University of Warsaw

Séminaire PLUME
19/11/2015
Lyon

Introduction

Deterministic automata on words are a central tool in automata theory:

- Polynomial algorithms for inclusion, complementation.
- Safe composition with games, trees.
- Solutions of the synthesis problem (verification).
- Easily implemented.

Problems :

- exponential state blow-up
- technical constructions (Safra)

Can we weaken the notion of determinism while preserving some good properties?

Good-for-Games automata

Idea : Nondeterminism can be resolved without knowledge about the future.

Good-for-Games automata

Idea : Nondeterminism can be resolved without knowledge about the future.

Introduced independently in

- symbolic representation (Henzinger, Piterman '06) $\rightarrow$ simplification
- quantitative models (Colcombet '09) $\rightarrow$ replace determinism

Applications

- synthesis
- branching time verification
- tree languages (Boker, K, Kupferman, S '12)

Evaluating a game

Finite alphabets I for inputs and O for outputs.
Synthesis : design a system responding to environment, while satisfying a constraint $\varphi \subseteq (IO)^\omega$ (regular language).

Environment: I_1
System:

.

Evaluating a game

Finite alphabets I for inputs and O for outputs.
Synthesis : design a system responding to environment, while satisfying a constraint $\varphi \subseteq (IO)^\omega$ (regular language).

Environment: I_1
System: O_1

.

Evaluating a game

Finite alphabets I for inputs and O for outputs.
Synthesis : design a system responding to environment, while satisfying a constraint $\varphi \subseteq (IO)^\omega$ (regular language).

Environment: I_1 I_2
System: O_1

.

Evaluating a game

Finite alphabets I for inputs and O for outputs.
Synthesis : design a system responding to environment, while satisfying a constraint $\varphi \subseteq (IO)^\omega$ (regular language).

Environment: I_1 I_2
System: O_1 O_2

.

Evaluating a game

Finite alphabets I for inputs and O for outputs.
Synthesis : design a system responding to environment, while satisfying a constraint $\varphi \subseteq (IO)^\omega$ (regular language).

Environment: I_1 I_2 I_3
System: O_1 O_2

.

Evaluating a game

Finite alphabets I for inputs and O for outputs.
Synthesis : design a system responding to environment, while satisfying a constraint $\varphi \subseteq (IO)^\omega$ (regular language).

Environment: I_1 I_2 I_3
System: O_1 O_2 O_3

.

Evaluating a game

Finite alphabets I for inputs and O for outputs.
Synthesis : design a system responding to environment, while satisfying a constraint $\varphi \subseteq (IO)^\omega$ (regular language).

Environment: $I_1 \quad I_2 \quad I_3 \quad \cdots$
System: $O_1 \quad O_2 \quad O_3 \quad \cdots$

System wins iff $(I_1, O_1), (I_2, O_2), (I_3, O_3), \ldots \models \varphi$.

.

Evaluating a game

Finite alphabets I for inputs and O for outputs.
Synthesis : design a system responding to environment, while satisfying a constraint $\varphi \subseteq (IO)^\omega$ (regular language).

Environment: $I_1 \quad I_2 \quad I_3 \quad \cdots$
System: $\quad O_1 \quad O_2 \quad O_3 \quad \cdots$

System wins iff $(I_1, O_1), (I_2, O_2), (I_3, O_3), \ldots \models \varphi$.

Church's problem: Can the system win ? If yes give strategy.

Evaluating a game

Finite alphabets I for inputs and O for outputs.
Synthesis : design a system responding to environment, while satisfying a constraint $\varphi \subseteq (IO)^\omega$ (regular language).

Environment: $I_1 \quad I_2 \quad I_3 \quad \cdots$
System: $O_1 \quad O_2 \quad O_3 \quad \cdots$

System wins iff $(I_1, O_1), (I_2, O_2), (I_3, O_3), \ldots \models \varphi$.

Church's problem: Can the system win ? If yes give strategy.

Classical approach: $\varphi \rightsquigarrow \mathcal{A}_{det}$ then solve game on $\mathcal{A}_{det}$.
2EXP blow-up for φ in LTL

Evaluating a game

Finite alphabets I for inputs and O for outputs.
Synthesis : design a system responding to environment, while satisfying a constraint $\varphi \subseteq (IO)^\omega$ (regular language).

Environment: $I_1 \quad I_2 \quad I_3 \quad \cdots$
System: $\quad O_1 \quad O_2 \quad O_3 \quad \cdots$

System wins iff $(I_1, O_1), (I_2, O_2), (I_3, O_3), \ldots \models \varphi$.

Church's problem: Can the system win ? If yes give strategy.

Classical approach: $\varphi \rightsquigarrow \mathcal{A}_{det}$ then solve game on $\mathcal{A}_{det}$.
2EXP blow-up for φ in LTL

Wrong approach: $\varphi \rightsquigarrow \mathcal{A}_{non-det}$: no player can guess the future.

A trivial synthesis example

Trivial instance of the synthesis problem:

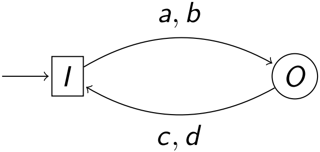
- $I = \{a, b\}$, $O = \{c, d\}$
- $\varphi = (IO)^\omega$
- Synthesis possible (no wrong answer !)

A trivial synthesis example

Trivial instance of the synthesis problem:

- $I = \{a, b\}$, $O = \{c, d\}$
- $\varphi = (IO)^\omega$
- Synthesis possible (no wrong answer !)

$\mathcal{A}_{det}$ (*safety*):

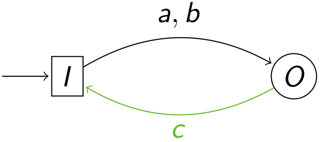


A trivial synthesis example

Trivial instance of the synthesis problem:

- $I = \{a, b\},\ O = \{c, d\}$
- $\varphi = (IO)^\omega$
- Synthesis possible (no wrong answer !)

$\mathcal{A}_{det}$ (*safety*):

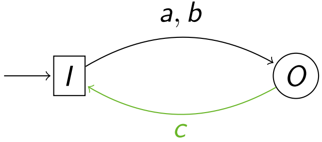


A trivial synthesis example

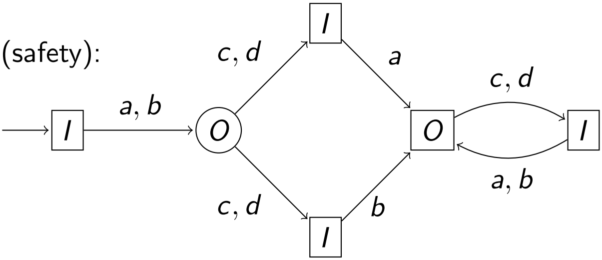
Trivial instance of the synthesis problem:

- $I = \{a, b\}$, $O = \{c, d\}$
- $\varphi = (IO)^\omega$
- Synthesis possible (no wrong answer !)

$\mathcal{A}_{det}$ (*safety*):



$\mathcal{A}_{non-det}$ (safety):

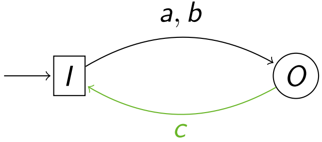


A trivial synthesis example

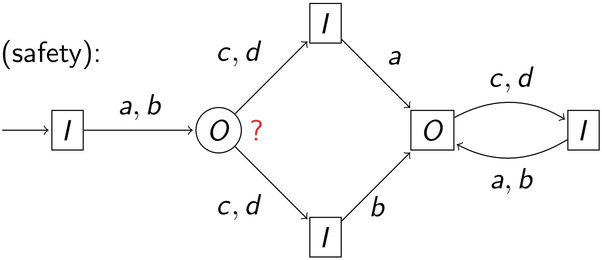
Trivial instance of the synthesis problem:

- $I = \{a, b\},\ O = \{c, d\}$
- $\varphi = (IO)^\omega$
- Synthesis possible (no wrong answer !)

$\mathcal{A}_{det}$ (*safety*):



$\mathcal{A}_{non-det}$ (safety):

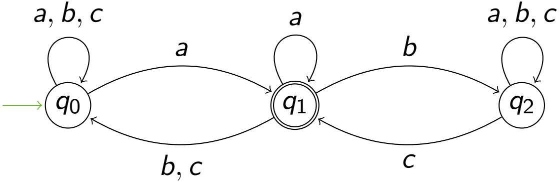


Definition of GFG via a game

$\mathcal{A}$ automaton on finite or infinite words.

Refuter plays letters:

GFG Prover: controls transitions

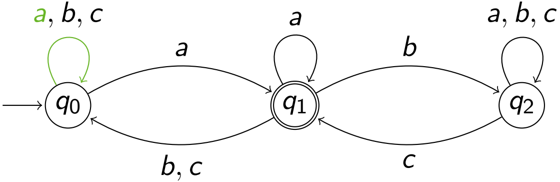


Definition of GFG via a game

$\mathcal{A}$ automaton on finite or infinite words.

Refuter plays letters: a

GFG Prover: controls transitions

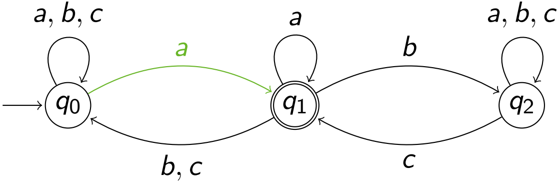


Definition of GFG via a game

$\mathcal{A}$ automaton on finite or infinite words.

Refuter plays letters: a a

GFG Prover: controls transitions

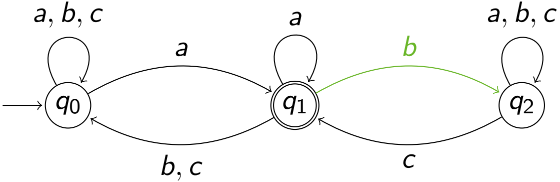


Definition of GFG via a game

$\mathcal{A}$ automaton on finite or infinite words.

Refuter plays letters: $a \;\; a \;\; b$

GFG Prover: controls transitions

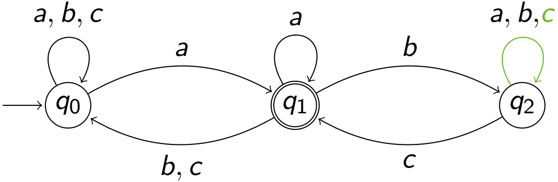


Definition of GFG via a game

$\mathcal{A}$ automaton on finite or infinite words.

Refuter plays letters: $a \ \ a \ \ b \ \ c$

GFG Prover: controls transitions

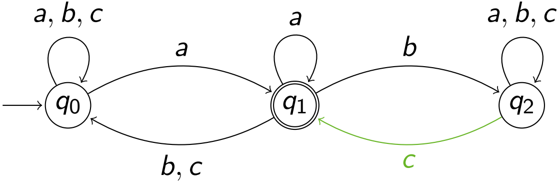


Definition of GFG via a game

$\mathcal{A}$ automaton on finite or infinite words.

Refuter plays letters: $a \quad a \quad b \quad c \quad c$

GFG Prover: controls transitions

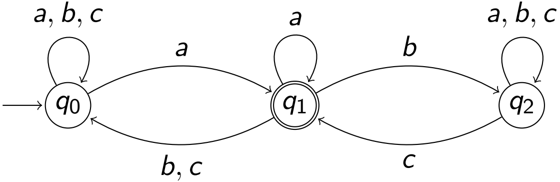


Definition of GFG via a game

$\mathcal{A}$ automaton on finite or infinite words.

Refuter plays letters: $a \;\; a \;\; b \;\; c \;\; c \; \ldots \quad = \; w$

GFG Prover: controls transitions



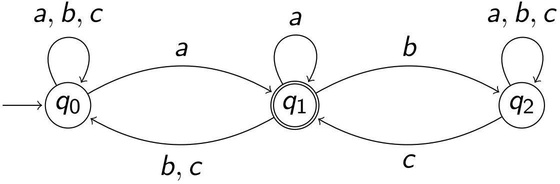
GFG Prover wins if: $w \in L \Rightarrow$ Run accepting.

Definition of GFG via a game

$\mathcal{A}$ automaton on finite or infinite words.

Refuter plays letters: $a \quad a \quad b \quad c \quad c \quad \ldots \quad = w$

GFG Prover: controls transitions



GFG Prover wins if: $w \in L \Rightarrow$ Run accepting.

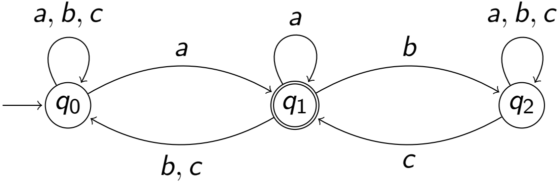
$\mathcal{A}$ **GFG** means that there is a strategy $\sigma : A^* \to Q$, for accepting words of $L(\mathcal{A})$.

Definition of GFG via a game

$\mathcal{A}$ automaton on finite or infinite words.

Refuter plays letters: $a \;\; a \;\; b \;\; c \;\; c \;\ldots \;\; = \; w$

GFG Prover: controls transitions



GFG Prover wins if: $w \in L \Rightarrow$ Run accepting.

$\mathcal{A}$ **GFG** means that there is a strategy $\sigma : A^* \to Q$, for accepting words of $L(\mathcal{A})$.

How close is this to determinism?

Why Good-for-games

Composing a game with an automaton:

Input:

- Game G with complex winning condition L.
  A alphabet of actions in G.
- Automaton $\mathcal{A}_L$ recognizing L, on alphabet A.
  Simple accepting condition C.

Output:

Game $\mathcal{A}_L \circ G$, with winning condition C.

Straightforward construction, arena of size $|\mathcal{A}_L| \cdot |G|$.

Goal: Simple winning condition $\rightsquigarrow$ positional winning strategies

Why Good-for-games

Composing a game with an automaton:

Input:

- Game G with complex winning condition L.
  A alphabet of actions in G.
- Automaton $\mathcal{A}_L$ recognizing L, on alphabet A.
  Simple accepting condition C.

Output:

Game $\mathcal{A}_L \circ G$, with winning condition C.
Straightforward construction, arena of size $|\mathcal{A}_L| \cdot |G|$.

Goal: Simple winning condition $\rightsquigarrow$ positional winning strategies

Theorem (Sound Composition)

$\mathcal{A}_L$ is **GFG** *if and only if*
for all G with condition L, $\mathcal{A}_L \circ G$ has same winner as G.

Some properties of GFG automata

GFG Automata:

- "$\mathcal{A} \subseteq \mathcal{B}$?": in **P** if $\mathcal{A}$ GFG (**PSPACE**-complete for ND)
- But Complementation $\sim$ Determinisation.
- Size of GFG strategy $\sigma \cong$ Size of deterministic automaton.

Some properties of GFG automata

GFG Automata:

- "$\mathcal{A} \subseteq \mathcal{B}$?": in **P** if $\mathcal{A}$ GFG (**PSPACE**-complete for ND)
- But Complementation $\sim$ Determinisation.
- Size of GFG strategy $\sigma \cong$ Size of deterministic automaton.

Theorem (Boker, K, Kupferman, S '12)

Let $\mathcal{A}$ be an automaton for $L \subseteq A^\omega$. Then the tree version of $\mathcal{A}$ recognizes $\{t :$ all branches of t are in $L\}$ if and only if $\mathcal{A}$ is ***GFG****.*

Some properties of GFG automata

GFG Automata:

- "$\mathcal{A} \subseteq \mathcal{B}$?": in **P** if $\mathcal{A}$ GFG (**PSPACE**-complete for ND)
- But Complementation $\sim$ Determinisation.
- Size of GFG strategy $\sigma \cong$ Size of deterministic automaton.

Theorem (Boker, K, Kupferman, S '12)

Let $\mathcal{A}$ be an automaton for $L \subseteq A^\omega$. Then the tree version of $\mathcal{A}$ recognizes $\{t : \text{all branches of } t \text{ are in } L\}$ if and only if $\mathcal{A}$ is **GFG**.

Theorem (Löding)

Let $\mathcal{A}$ be **GFG** *on finite words. Then $\mathcal{A}$ contains an equivalent deterministic automaton.*

Some properties of GFG automata

GFG Automata:

- "$\mathcal{A} \subseteq \mathcal{B}$?": in **P** if $\mathcal{A}$ GFG (**PSPACE**-complete for ND)
- But Complementation $\sim$ Determinisation.
- Size of GFG strategy $\sigma \cong$ Size of deterministic automaton.

Theorem (Boker, K, Kupferman, S '12)

Let $\mathcal{A}$ be an automaton for $L \subseteq A^\omega$. Then the tree version of $\mathcal{A}$ recognizes $\{t : \text{all branches of } t \text{ are in } L\}$ if and only if $\mathcal{A}$ is **GFG***.*

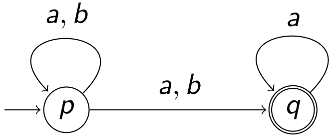
Theorem (Löding)

Let $\mathcal{A}$ be **GFG** *on finite words. Then $\mathcal{A}$ contains an equivalent deterministic automaton.*

What about infinite words ? *Colcombet's conjecture*: **GFG** $\approx$ Det.

An automaton that is not GFG

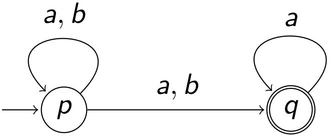
This automaton for $L = (a + b)^* a^\omega$ is not **GFG**:



Refuter strategy: play a until Eve goes in q, then play ba^ω.

An automaton that is not GFG

This automaton for $L = (a+b)^* a^\omega$ is not **GFG**:



Refuter strategy: play a until Eve goes in q, then play ba^ω.

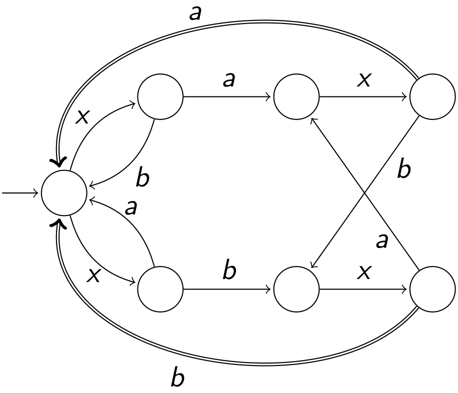
Fact

GFG *automata with condition* C *have same expressivity as deterministic automata with condition* C*.*

Therefore, **GFG** could improve succinctness but not expressivity.

A GFG Büchi example

Büchi condition: Run is accepting if infinitely many Büchi transitions are seen.



Language: $[(xa + xb)^*(xaxa + xbxb)]^\omega$

Determinization of Büchi GFG

Theorem

Let $\mathcal{A}$ a **GFG** *Büchi automaton. There exists a deterministic automaton $\mathcal{B}$ with $L(\mathcal{B}) = L(\mathcal{A})$ and $|\mathcal{B}| \leq |\mathcal{A}|^2$.*

Proof scheme:

- Brutal powerset determinisation,
- Use is as a guide to normalize $\mathcal{A}$.

Conclusion: the automaton can use itself as memory structure $\Rightarrow$ quadratic blow-up only.

Is it true for all ω-regular conditions?

The coBüchi jump

CoBüchi condition: must see finitely many rejecting states.

Fact (Miyano-Hayashi '84)

Nondeterministic CoBüchi automata are easier to determinise than Büchi ones: 2^n *instead of* $2^{n \log n}$ *and much simpler construction.*

Are CoBüchi **GFG** simpler to determinize than Büchi **GFG** ?

The coBüchi jump

CoBüchi condition: must see finitely many rejecting states.

Fact (Miyano-Hayashi '84)

Nondeterministic CoBüchi automata are easier to determinise than Büchi ones: 2^n instead of $2^{n \log n}$ and much simpler construction.

Are CoBüchi **GFG** simpler to determinize than Büchi **GFG** ? NO

Theorem

For all $n \geq 2$, there exists a language L_n on 3 letters such that

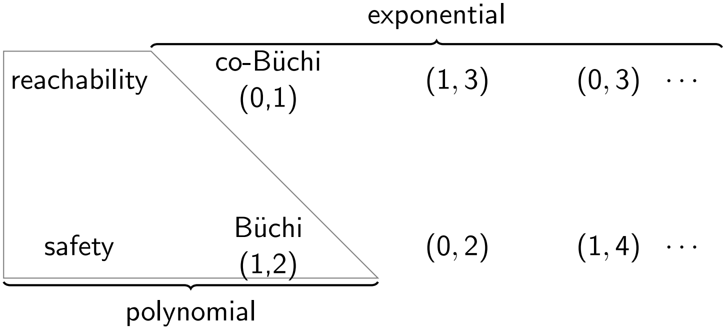
- *There is a n-state CoBüchi* **GFG** *automaton for L_n,*
- *any deterministic automaton for L_n has $\Omega(2^n)$ states.*

CoBüchi (and parity) **GFG** automata can provide both succinctness and sound behaviour with respect to games.

General picture

(i, j)-Parity condition: Each state has a color in $\{i, i+1, \ldots, j\}$.
Accepting runs: Maximal color occuring infinitely often is even.

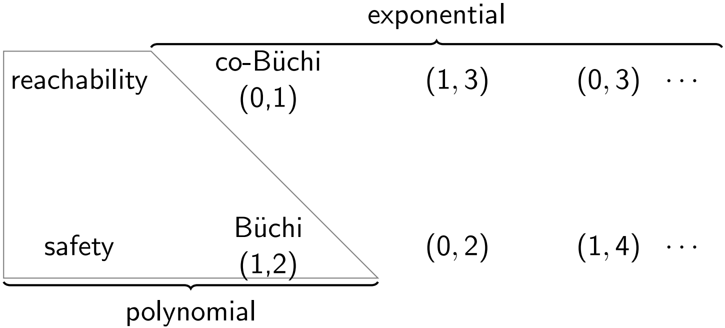
Blow-up GFG $\rightarrow$ Det:



General picture

(i,j)-Parity condition: Each state has a color in $\{i, i+1, \ldots, j\}$.
Accepting runs: Maximal color occuring infinitely often is even.

Blow-up GFG $\rightarrow$ Det:



Question: How practical are these **GFG** ?

Recognizing GFG automata

Question: Given an automaton $\mathcal{A}$, is it **GFG**?

Theorem

The complexity of deciding **GFG***-ness is in*

- *Upper bound:* **EXPTIME** *(even for* $(1,3)$*-parity)*
- **NP** *for Büchi automata*
- **P** *for coBüchi automata (surprising given blow-up result)*
- *at least as hard as solving parity games (***P** */* **NP** $\cap$ *co***NP***) for parity automata.*

Open Problems

- Is it in **P** for any fixed acceptance condition?
- Is it equivalent to parity games for arbitrary condition?

Summary and conclusion

Results

- **GFG** automata capture good properties of deterministic automata.
- Inclusion is in **P**, but Complementation $\sim$ Determinisation.
- Conditions Büchi and lower: **GFG** $\approx$ Deterministic.
- Conditions coBüchi and higher: exponential succinctness.
- Recognizing **GFG** coBüchi is in **P**.

Open Problems

- Can we build small **GFG** automata in a systematic way?
- Complexity of deciding **GFG**-ness for parity automata? (gap **P** vs **EXPTIME**)