Good-for-Games Automata.

Denis Kuperberg
joint work with Michał Skrzypczak

CNRS, ENS Lyon

01/02/2017
Oxford
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 ’13](#))
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: $O_1$

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: $I_1$ $I_2$
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**: $I_1 \ I_2 \ I_3 \ \cdots$

**System**: $O_1 \ O_2 \ O_3 \ \cdots$

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: $\varphi$; $A_{det}$ then solve game on $A_{det}$.

$2\text{EXP}$ blow-up for $\varphi$ in LTL

Wrong approach: $\varphi$; $A_{non-det}$: no player can guess the future.
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: $I_1 I_2 I_3 \cdots$

System: $O_1 O_2 O_3 \cdots$

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 $\varphi \subseteq (I O)^\omega$ (regular language).

Environment: $I_1 I_2 I_3 \cdots$

System: $O_1 O_2 O_3 \cdots$

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**: $\varphi \leadsto A_{det}$ then solve game on $A_{det}$.

2EXP blow-up for $\varphi$ 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: $I_1 \ I_2 \ I_3 \ \cdots$

System: $O_1 \ O_2 \ O_3 \ \cdots$

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: $\varphi \models A_{det}$ then solve game on $A_{det}$.

2EXP blow-up for $\varphi$ in LTL

Wrong approach: $\varphi \models A_{non-det}$: 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 !)

\( \mathcal{A}_{\text{det}} \) (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 !)

\( A_{det} \ (\text{safety}): \)

\[
\begin{array}{c}
\text{a, b} \\
\rightarrow \\
I \\
\rightarrow \\
O \\
\end{array}
\]

\[
\begin{array}{c}
\text{a, b} \\
\rightarrow \\
I \\
\rightarrow \\
O \\
\end{array}
\]

\[
\begin{array}{c}
\text{c} \\
\rightarrow \\
I \\
\rightarrow \\
O \\
\end{array}
\]

\[
\begin{array}{c}
\text{a, b} \\
\rightarrow \\
I \\
\rightarrow \\
O \\
\end{array}
\]

\[
\begin{array}{c}
\text{c} \\
\rightarrow \\
I \\
\rightarrow \\
O \\
\end{array}
\]
A trivial synthesis example

Trivial instance of the synthesis problem:

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

\[ A_{\text{det}} \text{ (safety):} \]

\[ A_{\text{non-det}} \text{ (safety):} \]
A trivial synthesis example

Trivial instance of the synthesis problem:

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

\[ A_{det} \text{ (safety):} \]

\[ A_{non-det} \text{ (safety):} \]
Definition of GFG via a game

A automaton on finite or infinite words.

Environment plays letters:

GFG Prover: controls transitions

\[ q_0 \rightarrow a, q_1 \rightarrow a, q_2 \rightarrow a \]

\[ q_0 \rightarrow b, c, q_1 \rightarrow b, q_2 \rightarrow c \]

\[ q_0 \rightarrow a, b, c, q_1 \rightarrow a, q_2 \rightarrow a, b, c \]
Definition of GFG via a game

A automaton on finite or infinite words.

Environment plays letters: $a$

GFG Prover: controls transitions
Definition of GFG via a game

A automaton on finite or infinite words.

Environment plays letters: a a

GFG Prover: controls transitions
Definition of GFG via a game

A automaton on finite or infinite words.
Environment plays letters: $a\ a\ b$

GFG Prover: controls transitions

\begin{itemize}
  \item $q_0$
  \item $q_1$
  \item $q_2$
\end{itemize}
Definition of GFG via a game

A automaton on finite or infinite words.

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

GFG Prover: controls transitions

![Diagram of an automaton](image-url)
Definition of GFG via a game

A automaton on finite or infinite words.  
Environment plays letters: $a\ a\ b\ c\ c$  
GFG Prover: controls transitions
Definition of GFG via a game

\[ A \text{ automaton on finite or infinite words.} \]

\[ \text{Environment plays letters: } a \ a \ b \ c \ c \ldots = w \]

\[ \text{GFG Prover: controls transitions} \]

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{c} q_1 \xrightarrow{a} q_0 \]

\[ \text{GFG Prover wins if: } w \in L \Rightarrow \text{Run accepting.} \]
A automaton on finite or infinite words. 

Environment plays letters: \( a \ a \ b \ c \ c \ldots = w \)

GFG Prover: controls transitions

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

A GFG means that there is a strategy \( \sigma : A^* \rightarrow Q \), for accepting words of \( L(A) \).
Definition of GFG via a game

A automaton on finite or infinite words.

Environment plays letters: $a\ a\ b\ c\ c\ \ldots = w$

GFG Prover: controls transitions

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

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

How close is this to determinism?
Why Good-for-games

Composing a game with an automaton:

Input:
- Game $G$ with complex winning condition $L$.
  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 $\leadsto$ positional winning strategies
Composing a game with an automaton:

**Input:**
- Game $G$ with complex winning condition $L$. 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 $\leadsto$ 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 $G$. 

**Why Good-for-games**
Some properties of GFG automata

GFG Automata:

- “\( A \subseteq B? \)”: in P if \( B \) GFG (PSPACE-complete for ND)
- But Complementation \( \sim \) Determinisation.
- Size of GFG strategy \( \sigma \approx \) Size of deterministic automaton.
Some properties of GFG automata

GFG Automata:

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

Theorem (Boker, K, Kupferman, S ’13)

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

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

**Theorem (Boker, K, Kupferman, S ’13)**

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:**

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

**Theorem (Boker, K, Kupferman, S ’13)**

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

**Theorem (Löding)**

Let $A$ be GFG on finite words. Then $A$ contains an equivalent deterministic automaton.

What about infinite words? **Colcombet’s conjecture:** GFG $\cong$ Det.
This automaton for \( L = (a + b)^* a^\omega \) is not GFG:

Environment strategy: play \( a \) until GFG Prover goes in \( q \), then play \( ba^\omega \).
An automaton that is not GFG

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

Environment strategy: play $a$ until GFG Prover goes in $q$, then play $ba^\omega$.

Fact

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

Therefore, GFG could improve succinctness but not expressivity.
Büchi condition: Run is accepting if infinitely many Büchi transitions are seen.

Language: \[ ((xa + xb)^* (xaxa + xbxb))^\omega \]
Theorem (K, Skrzypczak ’15)

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

Proof scheme:

▶ Brutal powerset determinisation,
▶ Use is as a guide to normalize $A$.

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

Is it true for all $\omega$-regular conditions?
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 \textit{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 (K, Skrzypczak '15)**

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

\begin{itemize}
  \item There is a $n$-state CoBüchi **GFG** automaton for $L_n$,
  \item any deterministic automaton for $L_n$ has $\Omega(2^n)$ states.
\end{itemize}\n
CoBüchi (and parity) **GFG** automata can provide both succinctness and sound behaviour with respect to games.*
**(i, j)-Parity condition:** Each state has a color in \(\{i, i + 1, \ldots, j\}\).

*Accepting runs:* Maximal color occurring infinitely often is even.

**Blow-up GFG \rightarrow Det:**

- **Reachability**
  - **Safety**
    - **Büchi**
      - \((1, 2)\)
    - **Co-Büchi**
      - \((0, 1)\)
  - **Polynomial**
    - **Exponential**
      - \((1, 3)\)
      - \((0, 3)\)
      - \(\cdots\)
General picture

\((i, j)\)-Parity condition: Each state has a color in \(\{i, i + 1, \ldots, j\}\).

Accepting runs: Maximal color occurring infinitely often is even.

Blow-up GFG → Det:

<table>
<thead>
<tr>
<th></th>
<th>exponential</th>
</tr>
</thead>
<tbody>
<tr>
<td>reachability</td>
<td>co-(\text{Büchi})</td>
</tr>
<tr>
<td>((0, 1))</td>
<td>((1, 3))</td>
</tr>
<tr>
<td>((1, 2))</td>
<td>((0, 2))</td>
</tr>
</tbody>
</table>

polynomial

Question: How practical are these GFG?
Recognizing GFG automata

**Question:** Given an automaton $A$, is it GFG?

**Theorem (K, Skrzypczak ’15)**

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 ($P / NP \cap coNP$) 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 $\mathbb{P}$, but Complementation $\sim$ Determinisation.
- Conditions Büchi and lower: **GFG** $\approx$ Deterministic.
- Conditions coBüchi and higher: exponential succinctness.
- Recognizing **GFG** coBüchi is in $\mathbb{P}$.

Open Problems

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