Good-for-Games Automata.

Denis Kuperberg    Michał Skrzypczak

TUM Munich and University of Warsaw

GT ALGA
12/04/2016
Marseille
**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**)
  → **simplification**
- quantitative models (**Colcombet ’09**) → **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: $O_1$
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: $O_1$ $O_2$
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 $\varphi \subseteq (I,O)^\omega$ (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.
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**: $\varphi \leadsto A_{det}$ 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**: $\varphi \rightsquigarrow A_{det}$ then solve game on $A_{det}$.

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

**Wrong approach**: $\varphi \rightsquigarrow A_{non-det}$: no player can guess the future.
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!)

\[ \mathcal{A}_{det} \text{ (safety):} \]

![Diagram](attachment:image_url)
A trivial synthesis example

Trivial instance of the synthesis problem:

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

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

\[ I \rightarrow - \rightarrow O \]

\[ a, b \rightarrow \]

\[ c \rightarrow \]
A trivial synthesis example

Trivial instance of the synthesis problem:

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

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

$A_{\text{non-det}}$ (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 !)

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

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

A automaton on finite or infinite words.

Refuter plays letters:

GFG Prover: controls transitions
A automaton on finite or infinite words.

Refuter plays letters: \( a \)

GFG Prover: controls transitions

\[
\begin{array}{c}
q_0 \\
q_1 \\
q_2
\end{array}
\]

- \( a, b, c \)
- \( a \) from \( q_0 \) to \( q_1 \)
- \( a \) from \( q_1 \) to \( q_2 \)
- \( b \) from \( q_1 \) to \( q_2 \)
- \( c \) from \( q_2 \) to \( q_1 \)
- \( b, c \) from \( q_0 \) to \( q_1 \)
- \( a, b, c \) from \( q_2 \) to \( q_0 \)
A automaton on finite or infinite words.

Refuter plays letters: $a \ a$

GFG Prover: controls transitions
A automaton on finite or infinite words.
Refuter plays letters: $a$  $a$  $b$

GFG Prover: controls transitions
Definition of GFG via a game

A automaton on finite or infinite words.
Refuter plays letters: \( a \ a \ b \ c \)

GFG Prover: controls transitions
Definition of GFG via a game

A automaton on finite or infinite words.
Refuter plays letters: \( a \ a \ b \ c \ c \)
GFG Prover: controls transitions
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: \(w \in L \Rightarrow \text{Run accepting.}\)
Definition of GFG via a game

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.

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

GFG Prover: controls transitions

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

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

How close is this to determinism?
Composing a game with an automaton:

**Input:**
- Game $G$ with complex winning condition $L$. $A$ alphabet of actions in $G$.
- Automaton $A_L$ recognizing $L$, on alphabet $A$. Simple accepting condition $C$.

**Output:**
Game $A_L \circ G$, with winning condition $C$. Straightforward construction, arena of size $|A_L| \cdot |G|$.

**Goal:** Simple winning condition $\leadsto$ positional winning strategies
Composing a game with an automaton:

Input:
- Game $G$ with complex winning condition $L$.
  - $A$ alphabet of actions in $G$.
- Automaton $A_L$ recognizing $L$, on alphabet $A$.
  - Simple accepting condition $C$.

Output:
Game $A_L \circ G$, with winning condition $C$.
Straightforward construction, arena of size $|A_L| \cdot |G|$.

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

**Theorem (Sound Composition)**

$A_L$ is GFG if and only if
for all $G$ with condition $L$, $A_L \circ G$ has same winner as $G$. 
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.
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 $A$ 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 $A$ be an automaton for $L \subseteq A^\omega$. Then the tree version of $A$ recognizes \{ $t$: all branches of $t$ 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.
Some properties of GFG automata

**GFG Automata:**
- “\( \mathcal{A} \subseteq \mathcal{B}? \)”: in \( \mathbf{P} \) if \( \mathcal{B} \) GFG (\( \mathbf{PSPACE} \)-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 \( L \subseteq \mathcal{A}^\omega \). 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 \( \cong \) 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 \( ba^\omega \).
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 \( ba^\omega \).

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

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 $\omega$-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 (K, Skrzypczak ’15)

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.
**(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)
    - Polynomial
  - Co-Büchi (0,1)
    - Exponential
      - (1,3)
      - (0,3)
      - \( \ldots \)
      - (0,2)
      - (1,4)
      - \( \ldots \)
(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:

- reachability
  - Büchi (1,2)
  - co-Büchi (0,1)

- safety
  - Büchi (1,2)

- polynomial

- exponential
  - (0,1)
  - (1,3)
  - (0,3)

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

Open Problems

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