Finite-state Strategies in Delay Games

Martin Zimmermann

Saarland University

September 21st, 2017

GandALF 2017, Rome, Italy
Motivation

Two goals:

1. Lift the notion of finite-state strategies to delay games.
2. Present uniform framework for solving delay games (which yields finite-state strategies whenever possible).
Motivation

Two goals:

1. Lift the notion of finite-state strategies to delay games.
2. Present uniform framework for solving delay games (which yields finite-state strategies whenever possible).

Questions:

- What are delay games?
- Why are finite-state strategies important?
- Why do we need a uniform framework?
In this talk, a game is given by an \( \omega \)-language \( L \subseteq (\Sigma_I \times \Sigma_O)^\omega \).

**Example**

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]
In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

**Example**

$$(\alpha(0)) (\alpha(1)) \cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i$$

$I$:  
$b$  

$O$:  

---

Martin Zimmermann  
Saarland University  
Finite-state Strategies in Delay Games
In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

**Example**

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\[I: \ b\]
\[O: \ a\]
In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

**Example**

$\begin{pmatrix} \alpha(0) \\ \beta(0) \end{pmatrix} \begin{pmatrix} \alpha(1) \\ \beta(1) \end{pmatrix} \cdots \in L$, if $\beta(i) = \alpha(i + 2)$ for every $i$

$I$: $b \ a$

$O$: $a$
In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

Example

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\ldots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

$I$: $b$ $a$

$O$: $a$ $a$
In this talk, a game is given by an ω-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

**Example**

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

$I$: $b$ $a$ $b$

$O$: $a$ $a$
Delay Games

In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

Example

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

$I$: $b$ $a$ $b$ $\ldots$

$O$: $a$ $a$ $\ldots$

$I$ wins
Delay Games

In this talk, a game is given by an \( \omega \)-language \( L \subseteq (\Sigma_I \times \Sigma_O)^\omega \).

**Example**

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\(I:\ b\ a\ b\ \cdots\)

\(O:\ a\ a\ \cdots\)

\(I\ wins\)

In a delay game, Player \(O\) may delay her moves to gain a lookahead on Player \(I\)'s moves.
Delay Games

In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

Example

$$
\begin{pmatrix}
\alpha(0)
\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1)
\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
$$

$\begin{array}{llllll}
I: & b & a & b & \cdots & I: & b \\
O: & a & a & \cdots & O: & \\
\end{array}$

$I$ wins

In a delay game, Player $O$ may delay her moves to gain a lookahead on Player $I$’s moves.
Delay Games

In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

Example

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\[
I: \ b \ a \ b \ \cdots \ \ I: \ b \ a \\
O: \ a \ a \ \cdots \ \ O: \\
\]

$I$ wins

In a delay game, Player $O$ may delay her moves to gain a lookahead on Player $I$'s moves.
In this talk, a game is given by an \( \omega \)-language \( L \subseteq (\Sigma_I \times \Sigma_O)\omega \).

**Example**

\[
\left(\begin{array}{c}
\alpha(0) \\
\beta(0)
\end{array}\right) \left(\begin{array}{c}
\alpha(1) \\
\beta(1)
\end{array}\right) \cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\begin{align*}
I & : b \ a \ b \ \cdots & I & : b \ a \ b \\
O & : a \ a \ \cdots & O & : \\
\end{align*}

\( I \) wins

In a delay game, Player \( O \) may delay her moves to gain a lookahead on Player \( I \)'s moves.
Delay Games

In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

Example

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

$$I: \quad b \quad a \quad b \quad \cdots \quad I: \quad b \quad a \quad b$$

$$O: \quad a \quad a \quad \cdots \quad O: \quad b$$

$I$ wins

In a delay game, Player $O$ may delay her moves to gain a lookahead on Player $I$’s moves.
In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

**Example**

\[
\begin{pmatrix}
  \alpha(0) \\
  \beta(0)
\end{pmatrix}
\begin{pmatrix}
  \alpha(1) \\
  \beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\begin{align*}
I: & \quad b \quad a \quad b \quad \cdots \\
O: & \quad a \quad a \quad \cdots
\end{align*}

\begin{align*}
I: & \quad b \quad a \quad b \quad b \\
O: & \quad b
\end{align*}

$I$ wins

In a delay game, Player $O$ may delay her moves to gain a lookahead on Player $I$’s moves.
In this talk, a game is given by an \( \omega \)-language \( L \subseteq (\Sigma_I \times \Sigma_O)^\omega \).

**Example**

\[
\begin{pmatrix} \alpha(0) \\ \beta(0) \end{pmatrix} \begin{pmatrix} \alpha(1) \\ \beta(1) \end{pmatrix} \cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\[
\text{I: } b \ a \ b \ b \cdots \quad \text{I: } b \ a \ b \ b \\
\text{O: } a \ a \ \cdots \quad \text{O: } b \ b
\]

**I wins**

In a delay game, Player \( O \) may delay her moves to gain a lookahead on Player \( I \)’s moves.
In this talk, a game is given by an \( \omega \)-language \( L \subseteq (\Sigma_I \times \Sigma_O)^\omega \).

**Example**

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\( I: \ b\ a\ b\ \cdots\ \ \ \ I: \ b\ a\ b\ b\ a\)

\( O: \ a\ a\ \cdots\ \ \ \ O: \ b\ b\)

**I wins**

In a delay game, Player \( O \) may delay her moves to gain a lookahead on Player \( I \)'s moves.
In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

**Example**

$\left(\begin{array}{c} \alpha(0) \\ \beta(0) \end{array}\right) \left(\begin{array}{c} \alpha(1) \\ \beta(1) \end{array}\right) \cdots \in L$, if $\beta(i) = \alpha(i + 2)$ for every $i$

$I$: $b$ $a$ $b$ $\cdots$  
$O$: $a$ $a$ $\cdots$

$I$ wins

In a delay game, Player $O$ may delay her moves to gain a lookahead on Player $I$’s moves.
Delay Games

In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

Example

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\begin{align*}
I: & \quad b \ a \ b \ \cdots \\
O: & \quad a \ a \ \cdots
\end{align*}

\begin{align*}
I: & \quad b \ a \ b \ b \ a \ b \\
O: & \quad b \ b \ a
\end{align*}

$I$ wins

In a delay game, Player $O$ may delay her moves to gain a lookahead on Player $I$’s moves.
In this talk, a game is given by an \( \omega \)-language \( L \subseteq (\Sigma_I \times \Sigma_O)^\omega \).

**Example**

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\[
\begin{array}{cccccccc}
I: & b & a & b & \cdots & I: & b & a & b & b & a & b \\
O: & a & a & \cdots & O: & b & b & a & b \\
\end{array}
\]

I wins

In a delay game, Player O may delay her moves to gain a lookahead on Player I’s moves.
In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

**Example**

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\[
\begin{array}{c}
I: \ b\ a\ b\ \cdots \\
O: \ a\ a\ \cdots
\end{array}
\quad
\begin{array}{c}
I: \ b\ a\ b\ b\ a\ b\ a\ a \\
O: \ b\ b\ a\ b
\end{array}
\]

I wins

In a delay game, Player O may delay her moves to gain a lookahead on Player I’s moves.
Delay Games

In this talk, a game is given by an $\omega$-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.

Example

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\begin{align*}
I: & \quad b \ a \ b \ \cdots \quad I: \quad b \ a \ b \ b \ a \ b \ a \\
O: & \quad a \ a \ \cdots \quad O: \quad b \ b \ a \ b \ a
\end{align*}

$I$ wins

In a delay game, Player $O$ may delay her moves to gain a lookahead on Player $I$’s moves.
Delay Games

In this talk, a game is given by an \( \omega \)-language \( L \subseteq (\Sigma_I \times \Sigma_O)^\omega \).

Example

\[
\begin{pmatrix} \alpha(0) \\ \beta(0) \end{pmatrix} \begin{pmatrix} \alpha(1) \\ \beta(1) \end{pmatrix} \cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\[I: \quad b \quad a \quad b \quad \cdots \quad I: \quad b \quad a \quad b \quad b \quad a \quad b \quad a \quad \cdots \]

\[O: \quad a \quad a \quad \cdots \quad O: \quad b \quad b \quad a \quad b \quad a \quad \cdots \]

\(I\) wins \hspace{1cm} \(O\) wins

In a delay game, Player \(O\) may delay her moves to gain a lookahead on Player \(I\)'s moves.
Hosch & Landweber ('72): $\omega$-regular delay games with respect to constant delay solvable.
Some History (1/2)

- **Hosch & Landweber (’72)**: \( \omega \)-regular delay games with respect to constant delay solvable.

- **Holtmann, Kaiser & Thomas (’10)**: Solving parity delay games in \( 2\text{ExpTime} \), doubly-exponential lookahead sufficient.
Some History (1/2)

- **Hosch & Landweber (’72):** $\omega$-regular delay games with respect to constant delay solvable.

- **Holtmann, Kaiser & Thomas (’10):** Solving parity delay games in $2\text{ExpTime}$, doubly-exponential lookahead sufficient.

- **Fridman, Löding & Z. (’11):** Nothing non-trivial is solvable for $\omega$-contextfree delay games, unbounded lookahead necessary.
Klein & Z. ('15): Solving parity delay games is $\text{ExpTime}$-complete, exponential lookahead sufficient and necessary.
Klein & Z. ('15): Solving parity delay games is \( \text{ExpTime} \)-complete, exponential lookahead sufficient and necessary.

Z. ('15): Max-regular delay games with respect to constant delay solvable, unbounded lookahead necessary.
Some History (2/2)

- **Klein & Z. (’15)**: Solving parity delay games is $\text{ExpTime}$-complete, exponential lookahead sufficient and necessary.

- **Z. (’15)**: Max-regular delay games with respect to constant delay solvable, unbounded lookahead necessary.

- **Klein & Z. (’16)**: Solving LTL delay games is $3\text{ExpTime}$-complete, triply-exponential lookahead sufficient and necessary.
Some History (2/2)

- **Klein & Z. ('15)**: Solving parity delay games is $\text{ExpTime}$-complete, exponential lookahead sufficient and necessary.

- **Z. ('15)**: Max-regular delay games with respect to constant delay solvable, unbounded lookahead necessary.

- **Klein & Z. ('16)**: Solving LTL delay games is $3\text{ExpTime}$-complete, triply-exponential lookahead sufficient and necessary.

- **Z. ('17)**: Solving cost-parity delay games is $\text{ExpTime}$-complete, exponential lookahead sufficient and necessary.
Some History (2/2)

- **Klein & Z. ('15)**: Solving parity delay games is $\text{ExpTime}$-complete, exponential lookahead sufficient and necessary.

- **Z. ('15)**: Max-regular delay games with respect to constant delay solvable, unbounded lookahead necessary.

- **Klein & Z. ('16)**: Solving LTL delay games is $3\text{ExpTime}$-complete, triply-exponential lookahead sufficient and necessary.

- **Z. ('17)**: Solving cost-parity delay games is $\text{ExpTime}$-complete, exponential lookahead sufficient and necessary.

All recent (positive) results use variations of the same proof idea.
A strategy in an infinite game is a map $\sigma: \Sigma_i^* \to \Sigma_O$, i.e., not necessarily finitely representable.
A strategy in an infinite game is a map $\sigma : \Sigma_i^* \rightarrow \Sigma_O$, i.e., not necessarily finitely representable. A finite-state strategy is implemented by a finite automaton with output, and therefore finitely represented.

Example

$$w \mapsto |w|_a \mod 2$$
Finite-state positional strategies are crucial in many applications of infinite games, e.g.:

- In reactive synthesis, a finite-state winning strategy is a correct-by-construction controller.
- (Modern proofs of) Rabin’s theorem rely on positional determinacy of parity games.
- In general, the existence of finite-state strategies enables the application of infinite games.
- Determining the memory requirements is one of the most fundamental tasks for a class of games.
Disclaimer: We focus here on constant delay!

- A strategy in a delay game is still a map $\sigma : \Sigma_i^* \rightarrow \Sigma_O$.
- So, the classical definition is still applicable.
- By “hardcoding” constant lookahead into the rules of the game, finite-state winning strategies are computable.
**Disclaimer:** We focus here on constant delay!

- A strategy in a delay game is still a map $\sigma : \Sigma_i^* \rightarrow \Sigma_O$.
- So, the classical definition is still applicable.
- By “hardcoding” constant lookahead into the rules of the game, finite-state winning strategies are computable.
- However, this notion does not distinguish “past” and “future”.
Example

\[ L = \left\{ \left( \begin{array}{c} \alpha \\ \alpha \end{array} \right) \mid \alpha \in \{0, 1\}^\omega \right\} \]
Example

$L = \left\{ \begin{pmatrix} \alpha \\ \alpha \end{pmatrix} \mid \alpha \in \{0, 1\}^\omega \right\}$

I:  a  a  b  a  a  b  b
O:  a  a  b  a  a  b  b

Requires $2d$ memory states with constant lookahead $d$. 
A (Cautionary) Example

Example

\[ L = \left\{ \begin{pmatrix} \alpha \\ \alpha \end{pmatrix} \mid \alpha \in \{0, 1\}^\omega \right\} \]

\[ \begin{array}{cccccccccc}
I: & a & a & b & a & a & b & b & b & a & a & a & b \\
O: & a & a & b & a & a & b & b & b & \end{array} \]
Example

\[ L = \{ \left( \begin{array}{c} \alpha \\ \alpha \end{array} \right) \mid \alpha \in \{0, 1\}^{\omega} \} \]

I: 
\[
\begin{array}{ccccccccccc}
\text{a} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{b} & \text{b} & \text{b} & \text{a} & \text{a} & \text{a} & \text{b} \\
\end{array}
\]

O: 
\[
\begin{array}{ccccccccccc}
\text{a} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} \\
\end{array}
\]

- Requires \(2^d\) memory states with constant lookahead \(d\).
Distinguishing between past and future: block games

- Fix a block length $d > 0$.
- Player $I$ picks blocks $\overline{a_i} \in \Sigma_I^d$.
- Player $O$ picks blocks $\overline{b_i} \in \Sigma_O^d$.
- Player $O$ wins, if $(\overline{a_0a_1a_2\ldots}) \in L$
- To account for (constant) lookahead, Player 1 is one move ahead.
Distinguishing between past and future: block games

- Fix a block length $d > 0$.
- Player $I$ picks blocks $\overline{a_i} \in \Sigma^d_I$.
- Player $O$ picks blocks $\overline{b_i} \in \Sigma^d_O$.
- Player $O$ wins, if $\left(\frac{a_0 a_1 a_2 \cdots}{b_0 b_1 b_2 \cdots}\right) \in L$
- To account for (constant) lookahead, Player 1 is one move ahead.

**Example**

$$\left(\begin{array}{c}
\alpha(0) \\
\beta(0)
\end{array}\right) \left(\begin{array}{c}
\alpha(1) \\
\beta(1)
\end{array}\right) \cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i$$

$I:$

$O:$
Distinguishing between past and future: block games

- Fix a block length $d > 0$.
- Player $I$ picks blocks $a_i \in \Sigma_I^d$.
- Player $O$ picks blocks $b_i \in \Sigma_O^d$.
- Player $O$ wins, if $(a_0 a_1 a_2 \cdots) \in L$
- To account for (constant) lookahead, Player 1 is one move ahead.

Example

$$\begin{pmatrix} \alpha(0) \\ \beta(0) \end{pmatrix} \begin{pmatrix} \alpha(1) \\ \beta(1) \end{pmatrix} \cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i$$

- $I: b a$
- $O:$

Martin Zimmermann  Saarland University  Finite-state Strategies in Delay Games 10/17
Distinguishing between past and future: block games

- Fix a block length $d > 0$.
- Player $I$ picks blocks $\overline{a_i} \in \Sigma_I^d$.
- Player $O$ picks blocks $\overline{b_i} \in \Sigma_O^d$.
- Player $O$ wins, if $\left( \frac{a_0 a_1 a_2 \cdots}{b_0 b_1 b_2 \cdots} \right) \in L$.
- To account for (constant) lookahead, Player 1 is one move ahead.

Example

\[
\begin{pmatrix} \alpha(0) \\ \alpha(1) \end{pmatrix} \cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\[
\begin{array}{cccc}
I: & b & a & b & a \\
O: & & & & 
\end{array}
\]
Distinguishing between past and future: block games

- Fix a block length \( d > 0 \).
- Player \( I \) picks blocks \( \overline{a_i} \in \Sigma_I^d \).
- Player \( O \) picks blocks \( \overline{b_i} \in \Sigma_O^d \).
- Player \( O \) wins, if \((\overline{a_0a_1a_2\ldots}) \in L\).
- To account for (constant) lookahead, Player 1 is one move ahead.

**Example**

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\( I: \ b \ a \ b \ a \)

\( O: \ b \ a \)
Block Games

Distinguishing between past and future: block games

- Fix a block length $d > 0$.
- Player $I$ picks blocks $a_i \in \Sigma^d_I$.
- Player $O$ picks blocks $b_i \in \Sigma^d_O$.
- Player $O$ wins, if $\left( a_0 a_1 a_2 \cdots \right) \in L$
- To account for (constant) lookahead, Player 1 is one move ahead.

Example

$\left( \alpha(0) \right) \left( \alpha(1) \right) \cdots \in L$, if $\beta(i) = \alpha(i + 2)$ for every $i$

$I$: $b$ $a$ $b$ $a$ $b$ $a$

$O$: $b$ $a$
Distinguishing between past and future: block games

- Fix a block length $d > 0$.
- Player $I$ picks blocks $\overline{a_i} \in \Sigma_I^d$.
- Player $O$ picks blocks $\overline{b_i} \in \Sigma_O^d$.
- Player $O$ wins, if $\left( \frac{a_0 a_1 a_2 \cdots}{b_0 b_1 b_2 \cdots} \right) \in L$
- To account for (constant) lookahead, Player 1 is one move ahead.

Example

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

$I$: b a b a b a a

$O$: b a b a a
Distinguishing between past and future: block games

- Fix a block length $d > 0$.
- Player $I$ picks blocks $\overline{a_i} \in \Sigma_I^d$.
- Player $O$ picks blocks $\overline{b_i} \in \Sigma_O^d$.
- Player $O$ wins, if $(\overline{a_0a_1a_2\cdots}) \in L$
- To account for (constant) lookahead, Player 1 is one move ahead.

Example

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\text{I: } b\ a\ b\ a\ b\ a\ b\ a\ b\ b
\text{O: } b\ a\ b\ a\ a
Distinguishing between past and future: block games

- Fix a block length $d > 0$.
- Player $I$ picks blocks $\overline{a}_i \in \Sigma_I^d$.
- Player $O$ picks blocks $\overline{b}_i \in \Sigma_O^d$.
- Player $O$ wins, if $\left(\overline{a}_0 \overline{a}_1 \overline{a}_2 \cdots \right) \in L$
- To account for (constant) lookahead, Player 1 is one move ahead.

Example

$$\left(\begin{array}{c} \alpha(0) \\ \beta(0) \end{array}\right) \left(\begin{array}{c} \alpha(1) \\ \beta(1) \end{array}\right) \cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i$$

$I$: $b \ a \ b \ a \ b \ a \ b \ b$

$O$: $b \ a \ b \ a \ b \ a \ b \ b$
Distinguishing between past and future: block games

- Fix a block length \(d > 0\).
- Player \(I\) picks blocks \(\overline{a}_i \in \Sigma^d_I\).
- Player \(O\) picks blocks \(\overline{b}_i \in \Sigma^d_O\).
- Player \(O\) wins, if \((\overline{a}_0 \overline{a}_1 \overline{a}_2 \cdots) / (\overline{b}_0 \overline{b}_1 \overline{b}_2 \cdots) \in L\)
- To account for (constant) lookahead, Player 1 is one move ahead.

**Example**

\[
\begin{pmatrix}
  \alpha(0) \\
  \beta(0)
\end{pmatrix} \begin{pmatrix}
  \alpha(1) \\
  \beta(1)
\end{pmatrix} \cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\[I: \quad \begin{array}{cccccccc}
  b & a & b & a & b & a & b & b & a & b
\end{array}
\]

\[O: \quad \begin{array}{cccccccc}
  b & a & b & a & b & a & b & b
\end{array} \]
Distinguishing between past and future: block games

- Fix a block length $d > 0$.
- Player $I$ picks blocks $\overline{a_i} \in \Sigma_I^d$.
- Player $O$ picks blocks $\overline{b_i} \in \Sigma_O^d$.
- Player $O$ wins, if $(\overline{a_0a_1a_2\cdots}) \in L$
- To account for (constant) lookahead, Player 1 is one move ahead.

Example

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix}
\cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

$$I: \ b \ a \ b \ a \ b \ a \ b \ b \ a \ b$$

$$O: \ b \ a \ b \ a \ b \ a \ b \ b \ a \ b$$
Distinguishing between past and future: block games

- Fix a block length \( d > 0 \).
- Player \( I \) picks blocks \( \overline{a_i} \in \Sigma_I^d \).
- Player \( O \) picks blocks \( \overline{b_i} \in \Sigma_O^d \).
- Player \( O \) wins, if \( \left( \overline{a_0a_1a_2}\ldots \right) \in L \).
- To account for (constant) lookahead, Player 1 is one move ahead.

**Example**

\[
\begin{pmatrix}
\alpha(0) \\
\beta(0)
\end{pmatrix}
\begin{pmatrix}
\alpha(1) \\
\beta(1)
\end{pmatrix} \cdots \in L, \text{ if } \beta(i) = \alpha(i + 2) \text{ for every } i
\]

\( I: \ b \ a \ b \ a \ b \ a \ b \ b \ a \ b \ \cdots \)

\( O: \ b \ a \ b \ a \ b \ b \ a \ b \ b \ \cdots \)
A finite-state strategy in a block game reads blocks over $\Sigma_I$ and outputs blocks in $\Sigma_O$:

$$I: \overline{a_0} \quad \overline{a_1} \quad \ldots \ldots \quad \overline{a_{i-2}} \quad \overline{a_{i-1}} \quad \overline{a_i}$$

$$O: \overline{b_0} \quad \overline{b_1} \quad \ldots \ldots \quad \overline{b_{i-2}}$$

Note: Alphabet now exponential in block length!

But, we distinguish past and future.

In particular, state complexity only concerned with past.
A finite-state strategy in a block game reads blocks over $\Sigma_I$ and outputs blocks in $\Sigma_O$:

$L:$ $\overline{a_0}$ $\overline{a_1}$ $\ldots \ldots$ $\overline{a_{i-2}}$ $\overline{a_{i-1}}$ $\overline{a_i}$

$O:$ $\overline{b_0}$ $\overline{b_1}$ $\ldots \ldots$ $\overline{b_{i-2}}$

Note: Alphabet now exponential in block length!

But, we distinguish past and future. In particular, state complexity only concerned with past.
A finite-state strategy in a block game reads blocks over $\Sigma_I$ and outputs blocks in $\Sigma_O$:

$I$: $\overline{a_0}$ $\overline{a_1}$ $\ldots$ $\overline{a_{i-2}}$ $\overline{a_{i-1}}$ $\overline{a_i}$

$O$: $\overline{b_0}$ $\overline{b_1}$ $\ldots$ $\overline{b_{i-2}}$ $\overline{b_{i-1}}$

$q \rightarrow \lambda(q, \overline{a_{i-1}}, \overline{a_i})$
A finite-state strategy in a block game reads blocks over $\Sigma_I$ and outputs blocks in $\Sigma_O$:

\[
\begin{align*}
I: & \quad a_0 \quad a_1 \quad \ldots \quad a_{i-2} \quad a_{i-1} \quad a_i \\
O: & \quad b_0 \quad b_1 \quad \ldots \quad b_{i-2} \quad b_{i-1} = \lambda(q, a_{i-1}, a_i)
\end{align*}
\]

Note:
- Alphabet now exponential in block length!
- But, we distinguish past and future.
- In particular, state complexity only concerned with past.
Fix ω-automaton $A$ and a finite set $M$.

$s: Q^+ \rightarrow M$ is an aggregation for $A$, if for all runs $\rho = \pi_0\pi_1\pi_2\cdots$ and $\rho' = \pi'_0\pi'_1\pi'_2\cdots$ with $s(\pi_0)s(\pi_1)s(\pi_2)\cdots = s(\pi'_0)s(\pi'_1)s(\pi'_2)\cdots$: $\rho$ is accepting $\iff$ $\rho'$ is accepting.
Fix ω-automaton $\mathcal{A}$ and a finite set $M$.

$s : Q^+ \rightarrow M$ is an aggregation for $\mathcal{A}$, if for all runs $\rho = \pi_0\pi_1\pi_2 \cdots$ and $\rho' = \pi'_0\pi'_1\pi'_2 \cdots$ with $s(\pi_0)s(\pi_1)s(\pi_2) \cdots = s(\pi'_0)s(\pi'_1)s(\pi'_2) \cdots$: $\rho$ is accepting $\iff$ $\rho'$ is accepting.

\[
\begin{array}{cccccc}
\pi_0 & \pi_1 & \pi_2 & \pi_3 & \pi_4 & \cdots
\end{array}
\]
Fix $\omega$-automaton $\mathcal{A}$ and a finite set $M$.

$s: Q^+ \to M$ is an aggregation for $\mathcal{A}$, if for all runs $\rho = \pi_0\pi_1\pi_2\cdots$ and $\rho' = \pi'_0\pi'_1\pi'_2\cdots$ with $s(\pi_0)s(\pi_1)s(\pi_2)\cdots = s(\pi'_0)s(\pi'_1)s(\pi'_2)\cdots$: $\rho$ is accepting $\iff$ $\rho'$ is accepting.

Example $q_0 \rightarrow \cdots q_i \mapsto \max_{0 \leq j \leq i} \Omega(q_j)$ is an aggregation for a max-parity automaton with coloring $\Omega$.
Aggregations

- Fix $\omega$-automaton $\mathfrak{A}$ and a finite set $M$.
- $s : Q^+ \rightarrow M$ is an aggregation for $\mathfrak{A}$, if for all runs $\rho = \pi_0\pi_1\pi_2\cdots$ and $\rho' = \pi'_0\pi'_1\pi'_2\cdots$ with $s(\pi_0)s(\pi_1)s(\pi_2)\cdots = s(\pi'_0)s(\pi'_1)s(\pi'_2)\cdots$: $\rho$ is accepting $\iff$ $\rho'$ is accepting.

![Diagram](image-url)
Aggregations

- Fix $\omega$-automaton $A$ and a finite set $M$.
- $s : Q^+ \rightarrow M$ is an aggregation for $A$, if for all runs $\rho = \pi_0 \pi_1 \pi_2 \cdots$ and $\rho' = \pi'_0 \pi'_1 \pi'_2 \cdots$ with $s(\pi_0)s(\pi_1)s(\pi_2)\cdots = s(\pi'_0)s(\pi'_1)s(\pi'_2)\cdots$: $\rho$ is accepting $\iff$ $\rho'$ is accepting.

Example

$q_0 \cdots q_i \mapsto \max_{0 \leq j \leq i} \Omega(q_j)$ is an aggregation for a max-parity automaton with coloring $\Omega$. 
Every automaton $\mathcal{M}$ with input alphabet $Q$ and state set $\mathcal{M}$ computes an aggregation $s_{\mathcal{M}} : Q^+ \rightarrow \mathcal{M}$: $s_{\mathcal{M}}(\pi)$ is the state reached by $\mathcal{M}$ when processing $\pi$. 
Every automaton $\mathcal{M}$ with input alphabet $Q$ and state set $M$ computes an aggregation $s_{\mathcal{M}} : Q^+ \rightarrow M$: $s_{\mathcal{M}}(\pi)$ is the state reached by $\mathcal{M}$ when processing $\pi$.

Example

$q_0 \cdots q_i \mapsto \max_{0 \leq j \leq i} \Omega(q_j)$ computable by automaton with state set $\Omega(Q)$. 
Fix $\mathcal{A}$ recognizing winning condition $L(\mathcal{A}) \subseteq (\Sigma_I \times \Sigma_O)^\omega$ and let $s_M : Q^+ \rightarrow M$ be aggregation for $\mathcal{A}$ computed by some $M$.

- Define $x \equiv x'$ iff $x$ and $x'$ induce the same behavior in $\mathcal{A}$, i.e., the same state changes and the corresponding runs have the same $s_M$-value.
Abstract Block Games

Fix $\mathcal{A}$ recognizing winning condition $L(\mathcal{A}) \subseteq (\Sigma_I \times \Sigma_O)^\omega$ and let $s_M: Q^+ \rightarrow M$ be aggregation for $\mathcal{A}$ computed by some $M$.

- Define $x \equiv x'$ iff $x$ and $x'$ induce the same behavior in $\mathcal{A}$, i.e., the same state changes and the corresponding runs have the same $s_M$-value.

$x \in \Sigma^*$:
Fix $\mathcal{A}$ recognizing winning condition $\mathcal{L}(\mathcal{A}) \subseteq (\Sigma_I \times \Sigma_O)^\omega$ and let $s^*: Q^+ \rightarrow M$ be aggregation for $\mathcal{A}$ computed by some $\mathcal{M}$.

- Define $x \equiv x'$ iff $x$ and $x'$ induce the same behavior in $\mathcal{A}$, i.e., the same state changes and the corresponding runs have the same $s^*$-value.
Abstract Block Games

Fix $A$ recognizing winning condition $L(A) \subseteq (\Sigma_I \times \Sigma_O)^\omega$ and let $s_{\equiv M} : Q^+ \rightarrow M$ be aggregation for $A$ computed by some $M$.

- Define $x \equiv x'$ iff $x$ and $x'$ induce the same behavior in $A$, i.e., the same state changes and the corresponding runs have the same $s_{\equiv M}$-value.

\[
q_0 \\
q_1 \\
\vdots \\
q_n
\]
Fix $\mathcal{A}$ recognizing winning condition $L(\mathcal{A}) \subseteq (\Sigma_I \times \Sigma_O)^\omega$ and let $s_\mathcal{M} : Q^+ \rightarrow M$ be aggregation for $\mathcal{A}$ computed by some $\mathcal{M}$.

Define $x \equiv x'$ iff $x$ and $x'$ induce the same behavior in $\mathcal{A}$, i.e., the same state changes and the corresponding runs have the same $s_\mathcal{M}$-value.

\[ q_0 \quad \cdots \quad q_n \quad \rightarrow \quad q'_0 \quad \cdots \quad q'_n \]
Fix $\mathcal{A}$ recognizing winning condition $L(\mathcal{A}) \subseteq (\Sigma_I \times \Sigma_O)^\omega$ and let $s_M : Q^+ \rightarrow M$ be aggregation for $\mathcal{A}$ computed by some $\mathcal{M}$.

- Define $x \equiv x'$ iff $x$ and $x'$ induce the same behavior in $\mathcal{A}$, i.e., the same state changes and the corresponding runs have the same $s_M$-value.
Fix $\mathcal{A}$ recognizing winning condition $L(\mathcal{A}) \subseteq (\Sigma_I \times \Sigma_O)^\omega$ and let $s_M : Q^+ \rightarrow M$ be aggregation for $\mathcal{A}$ computed by some $M$.

- Define $x \equiv x'$ iff $x$ and $x'$ induce the same behavior in $\mathcal{A}$, i.e., the same state changes and the corresponding runs have the same $s_M$-value.

- $\equiv$ has index at most $2|Q|^2|M|$.
Abstract Block Games

The abstract block game is played as follows:

- Player $I$ picks equivalence classes $S_0 S_1 \cdots$.
- Player $O$ picks compatible sequence $(q_0, \ast)(q_1, m_1) \cdots$.

Player $O$ wins if $m_1 m_2 \cdots$ is aggregation of accepting run.

This is a delay-free Gale-Stewart game!

Automaton reconizing winning condition is (roughly) of size $O(\text{index}(\equiv))$. 
The abstract block game is played as follows:

- Player $I$ picks equivalence classes $S_0S_1\cdots$.
- Player $O$ picks compatible sequence $(q_0,\ast)(q_1, m_1)\cdots$.

$S_0$
The abstract block game is played as follows:
- Player $I$ picks equivalence classes $S_0 S_1 \cdots$.
- Player $O$ picks compatible sequence $(q_0, *) (q_1, m_1) \cdots$.

Player $O$ wins if $m_1 m_2 \cdots$ is aggregation of accepting run. This is a delay-free Gale-Stewart game! Automaton reconizing winning condition is (roughly) of size $O(\text{index}(\equiv))$. 
The abstract block game is played as follows:

- **Player I** picks equivalence classes $S_0 S_1 \cdots$.
- **Player O** picks compatible sequence $(q_0, *) (q_1, m_1) \cdots$.

Player O wins if $m_1 m_2 \cdots$ is an aggregation of an accepting run.

This is a delay-free Gale-Stewart game!

Automaton recognizing the winning condition is (roughly) of size $O(\text{index}(\equiv))$. 
Abstract Block Games

The abstract block game is played as follows:
- Player $P$ picks equivalence classes $S_0S_1\cdots$.
- Player $O$ picks compatible sequence $(q_0, *)(q_1, m_1)\cdots$.

\[
\begin{array}{c}
S_0 \\
\cup \\
\hline \\
q_1 \\
S_1
\end{array}
\]

Player $O$ wins if $m_1m_2\cdots$ is aggregation of accepting run.

This is a delay-free Gale-Stewart game!

Automaton recognizing winning condition is (roughly) of size $O(\text{index}(\equiv))$. 
Abstract Block Games

The abstract block game is played as follows:

- Player $I$ picks equivalence classes $S_0 S_1 \cdots$.
- Player $O$ picks compatible sequence $(q_0, *) (q_1, m_1) \cdots$.

$S_0 \cup S_1$

$q_I$
The abstract block game is played as follows:

- Player $I$ picks equivalence classes $S_0, S_1, \ldots$.
- Player $O$ picks compatible sequence $(q_0, \ast)(q_1, m_1) \ldots$.

Player $O$ wins if $m_1, m_2, \ldots$ is an aggregation of accepting run.

This is a delay-free Gale-Stewart game!

Automaton recognizing winning condition is (roughly) of size $O(\text{index}(\equiv))$.
The abstract block game is played as follows:

- Player $I$ picks equivalence classes $S_0S_1\cdots$.
- Player $O$ picks compatible sequence $(q_0,*)(q_1,m_1)\cdots$.

Player $O$ wins if $m_1m_2\cdots$ is an accepting run. This is a delay-free Gale-Stewart game! Automaton recognizing winning condition is (roughly) of size $O(\text{index}(\equiv))$. 

\begin{center}
\begin{tikzpicture}
  \node (S0) at (0,0) {$S_0$};
  \node (S1) at (1.5,0) {$S_1$};
  \node (S2) at (3,0) {$S_2$};
  \node (q1) at (1.5,-2) {$q_1$};
  \node (m1) at (1.5,-3) {$m_1$};
  \draw (S0) -- (S1);
  \draw (S1) -- (S2);
  \draw (q1) -- (m1) node[midway,above] {$\cup$};
  \draw (m1) -- (q1) node[midway,above] {$*$};
\end{tikzpicture}
\end{center}
Abstract Block Games

The abstract block game is played as follows:

- Player $I$ picks equivalence classes $S_0, S_1, \ldots$.
- Player $O$ picks compatible sequence $(q_0, \ast)(q_1, m_1) \ldots$.

Player $O$ wins if $m_1 m_2 \ldots$ is aggregation of accepting run.

This is a delay-free Gale-Stewart game!

Automaton reconizing winning condition is (roughly) of size $O(\text{index}(\equiv)).$
Abstract Block Games

The abstract block game is played as follows:

- Player \( I \) picks equivalence classes \( S_0 S_1 \cdots \).
- Player \( O \) picks compatible sequence \((q_0, *)(q_1, m_1)\cdots\).

\[
\begin{align*}
S_0 & \subseteq q^I \subseteq S_1 & S_1 & \subseteq q^I \subseteq S_2 \\
& \cup & & \cup \\
& \subseteq & & \subseteq \\
q_I & \xrightarrow{m_1} q_1
\end{align*}
\]
Abstract Block Games

The abstract block game is played as follows:

- Player $I$ picks equivalence classes $S_0 S_1 \cdots$.
- Player $O$ picks compatible sequence $(q_0, \ast)(q_1, m_1) \cdots$.

Player $O$ wins if $m_1 m_2 \cdots$ is an aggregation of accepting runs.

This is a delay-free Gale-Stewart game!

Automaton recognizing winning condition is (roughly) of size $O(\text{index}(\equiv))$. 

\begin{figure}
\centering
\begin{tikzpicture}
\node (S0) at (0,0) {$S_0$};
\node (S1) at (2,0) {$S_1$};
\node (S2) at (4,0) {$S_2$};
\node (q1) at (0,-1) {$q_1$};
\node (m1) at (0,-2) {$m_1$};
\node (q2) at (4,-2) {$q_2$};
\node (m2) at (4,-1) {$m_2$};
\draw (S0) -- (q1) -- (m1) -- (S1) -- (q1) -- (m2) -- (S2);
\end{tikzpicture}
\end{figure}
The abstract block game is played as follows:

- Player $I$ picks equivalence classes $S_0S_1\cdots$.
- Player $O$ picks compatible sequence $(q_0,*)(q_1,m_1)\cdots$.

Player $O$ wins if $m_1m_2\cdots$ is aggregation of accepting run.

This is a delay-free Gale-Stewart game!

Automaton recognizing winning condition is (roughly) of size $O(\text{index}(\equiv))$. 

Player $O$ wins if $m_1m_2\cdots$ is aggregation of accepting run.
Abstract Block Games

The abstract block game is played as follows:

- Player I picks equivalence classes $S_0 S_1 \cdots$.
- Player O picks compatible sequence $(q_0, *)(q_1, m_1) \cdots$.

\[ S_0 \cup S_1 \cup S_2 \]

Player O wins if $m_1 m_2 \cdots$ is aggregation of accepting run.

- This is a delay-free Gale-Stewart game!
- Automaton reconizing winning condition is (roughly) of size $O(\text{index}(\equiv))$. 
Main Theorem

Theorem
Let $A$ be an $\omega$-automaton, let $\mathcal{M}$ be an aggregation for $A$, and define $d = 2|Q|^2 \cdot |\mathcal{M}|$.

1. If Player $O$ wins the delay game with winning condition $L(A)$ for any lookahead, then she also wins the corresponding abstract block game.

2. If Player $O$ wins the abstract block game, then she also wins the block game with winning condition $L(A)$ and block size $d$.

3. Moreover, if she has a finite-state winning strategy for the abstract game, then she has a finite-state winning strategy of the same size for the block game.
**Main Theorem**

**Theorem**
Let $\mathcal{A}$ be an $\omega$-automaton, let $s_M$ be an aggregation for $\mathcal{A}$, and define $d = 2|Q|^2 \cdot |M|$.

1. If Player $O$ wins the delay game with winning condition $L(\mathcal{A})$ for any lookahead, then she also wins the corresponding abstract block game.

2. If Player $O$ wins the abstract block game, then she also wins the block game with winning condition $L(\mathcal{A})$ and block size $d$.

3. Moreover, if she has a finite-state winning strategy for the abstract game, then she has a finite-state winning strategy of the same size for the block game.

**Corollary**
Solving delay games equivalent to solving abstract block games and constant lookahead $2d$ is sufficient.
Conclusion

Also in the Paper:

1. Another type of aggregation suitable for quantitative acceptance conditions.

2. The same framework yields decidability and finite-state strategies for quantitative delay games \textit{w.r.t. constant lookahead}. 

Recall that automata implementing finite-state strategies in block games process blocks $\Rightarrow$ Exponentially-sized alphabets.

1. Implement transition and output function as transducers.

2. Upper and lower bounds on size in both models.

3. Tradeoffs between these models.
Conclusion

Also in the Paper:

1. Another type of aggregation suitable for quantitative acceptance conditions.
2. The same framework yields decidability and finite-state strategies for quantitative delay games \textit{w.r.t. constant} lookahead.

\textbf{Unpublished} (with Sarah Winter):
Recall that automata implementing finite-state strategies in block games process blocks $\Rightarrow$ Exponentially-sized alphabets.

1. Implement transition and output function as transducers.
2. Upper and lower bounds on size in both models.
3. Tradeoffs between these models.