Games with delay for automaton synthesis

Christof Löding
RWTH Aachen University, Germany

GandALF
Borca di Cadore, Dolomites, Italy
August 29–31, 2013
1. Automaton Synthesis from Specifications
   - Classical Setting
   - Delaying the Output
   - Beyond Finite Automata

2. Synthesis of Lookahead Delegators
Outline

1. Automaton Synthesis from Specifications
   - Classical Setting
   - Delaying the Output
   - Beyond Finite Automata

2. Synthesis of Lookahead Delegators
Motivation: Realizing Specifications

\[ \text{specification} \subseteq \{\text{inputs}\} \times \{\text{outputs}\} \]
Motivation: Realizing Specifications

specification \subseteq \{\text{inputs}\} \times \{\text{outputs}\}

\begin{align*}
\text{synthesize} \\
\text{input} \quad \rightarrow \quad \text{program } P \quad \rightarrow \quad \text{output}
\end{align*}
Motivation: Realizing Specifications

\[
\text{specification} \subseteq \{\text{inputs}\} \times \{\text{outputs}\}
\]

\[\text{synthesize}\]

\[
\text{input} \quad \rightarrow \quad \text{program } P \quad \rightarrow \quad \text{output}
\]

input

output
Motivation: Realizing Specifications

\[ \text{specification} \subseteq \{\text{inputs}\} \times \{\text{outputs}\} \]

synthesize

input \rightarrow \text{program } P \rightarrow \text{output}

input \quad a_0

output
Motivation: Realizing Specifications

\[ \text{specification} \subseteq \{\text{inputs}\} \times \{\text{outputs}\} \]

synthesize

input \rightarrow \text{program } P \rightarrow \text{output}

\begin{align*}
\text{input} & : a_0 \\
\text{output} & : b_0
\end{align*}

Games with delay for automaton synthesis · GandALF 2013
Motivation: Realizing Specifications

specification \subseteq \{ \text{inputs} \} \times \{ \text{outputs} \}

synthesize

input \rightarrow \text{program } P \rightarrow \text{output}

input \quad a_0 \ a_1

output \quad b_0
Motivation: Realizing Specifications

specification $\subseteq \{\text{inputs}\} \times \{\text{outputs}\}$

synthesize

input $\rightarrow$ program $P$ $\rightarrow$ output

input

output

$P$

$\begin{bmatrix} a_0 & a_1 \\ b_0 & b_1 \end{bmatrix}$
Motivation: Realizing Specifications

specification \subseteq \{\text{inputs}\} \times \{\text{outputs}\}

input \rightarrow \text{program } P \rightarrow \text{output}

input: \ a_0 \ a_1 \ a_2

output: \ b_0 \ b_1
Motivation: Realizing Specifications

 specification $\subseteq \{\text{inputs}\} \times \{\text{outputs}\}$

 input $\rightarrow$ program $P$ $\rightarrow$ output

$\begin{align*}
\text{input} & : \{a_0, a_1, a_2\} \\
\text{output} & : \{b_0, b_1, b_2\}
\end{align*}$
Motivation: Realizing Specifications

specification $\subseteq \{\text{inputs}\} \times \{\text{outputs}\}$

program $P$

input $a_0 \ a_1 \ a_2 \ a_3$

output $b_0 \ b_1 \ b_2$
Motivation: Realizing Specifications

specification \subseteq \{\text{inputs}\} \times \{\text{outputs}\}

synthesize

input \rightarrow \text{program } P \rightarrow \text{output}

input \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \end{bmatrix}_P

output \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \end{bmatrix}_P
Motivation: Realizing Specifications

specification $\subseteq \{\text{inputs}\} \times \{\text{outputs}\}$

synthesize

input $\rightarrow$ program $P$ $\rightarrow$ output

input $a_0 \ a_1 \ a_2 \ a_3 \ a_4$

output $b_0 \ b_1 \ b_2 \ b_3$
Motivation: Realizing Specifications

\[ \text{specification} \subseteq \{\text{inputs}\} \times \{\text{outputs}\} \]

\[ \text{synthesize} \]

\[ \text{input} \rightarrow \text{program } P \rightarrow \text{output} \]

input \[ \begin{array}{c}
\[ a_0 \ a_1 \ a_2 \ a_3 \ a_4 \]
\end{array} \]

output \[ \begin{array}{c}
\[ b_0 \ b_1 \ b_2 \ b_3 \ b_4 \]
\end{array} \]
Motivation: Realizing Specifications

specification ⊆ \{\text{inputs}\} \times \{\text{outputs}\}

input \rightarrow \text{program } P \rightarrow \text{output}

\begin{align*}
\text{input} & : a_0 \ a_1 \ a_2 \ a_3 \ a_4 \cdots \\
\text{output} & : b_0 \ b_1 \ b_2 \ b_3 \ b_4 \cdots \in \text{specification}
\end{align*}
Viewed as a Game

Two players I (input) and II (output) play letters from finite alphabets ($I$ and $J$) in alternation:

I

II
Viewed as a Game

Two players I (input) and II (output) play letters from finite alphabets ($I$ and $J$) in alternation:

\[
\begin{array}{c}
I & a_0 \\
II \end{array}
\]
Two players I (input) and II (output) play letters from finite alphabets ($I$ and $J$) in alternation:

I  $a_0$

II $b_0$
Two players I (input) and II (output) play letters from finite alphabets \((I \text{ and } J)\) in alternation:

I \quad a_0 \ a_1

II \quad b_0
Two players I (input) and II (output) play letters from finite alphabets ($I$ and $J$) in alternation:

\begin{align*}
\text{I} & \quad a_0 \ a_1 \\
\text{II} & \quad b_0 \ b_1
\end{align*}
Viewed as a Game

Two players I (input) and II (output) play letters from finite alphabets ($I$ and $J$) in alternation:

I

\[ a_0 \ a_1 \ a_2 \]

II

\[ b_0 \ b_1 \]
Viewed as a Game

Two players I (input) and II (output) play letters from finite alphabets \((I\) and \(J\)) in alternation:

\[
\begin{array}{ccc}
I & a_0 & a_1 & a_2 \\
II & b_0 & b_1 & b_2
\end{array}
\]
Viewed as a Game

Two players I (input) and II (output) play letters from finite alphabets ($I$ and $J$) in alternation:

$$
\begin{align*}
\text{I} & \quad a_0 & a_1 & a_2 & \cdots \\
\text{II} & \quad b_0 & b_1 & b_2
\end{align*}
$$
Viewed as a Game

Two players I (input) and II (output) play letters from finite alphabets \(I\) and \(J\) in alternation:

\[
\begin{align*}
I & : a_0 \ a_1 \ a_2 \cdots \\
II & : b_0 \ b_1 \ b_2 \cdots
\end{align*}
\]
Viewed as a Game

Two players I (input) and II (output) play letters from finite alphabets (I and J) in alternation:

I \quad a_0 \ a_1 \ a_2 \cdots

II \quad b_0 \ b_1 \ b_2 \cdots

Winning condition: II wins if the pair \((a_0 a_1 a_2 \cdots, b_0 b_1 b_2 \cdots)\) is in the relation given by the specification.
Two players I (input) and II (output) play letters from finite alphabets ($I$ and $J$) in alternation:

$I \quad a_0 \ a_1 \ a_2 \cdots$

$II \quad b_0 \ b_1 \ b_2 \cdots$

Winning condition: II wins if the pair $(a_0a_1a_2\cdots, b_0b_1b_2\cdots)$ is in the relation given by the specification.

The desired program $P$ now corresponds to a winning strategy for player output.

Finite automaton solution: $P$ is a finite state machine $(S, I, s_0, \delta, f)$ with output function $f : S \times I \to J$. 

Viewed as a Game
Example

Alphabet \( \{0, 1\} \) for both players.

Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Example

Alphabet \{0, 1\} for both players.

Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1

Finite automaton solution:
Example

Alphabet \{0, 1\} for both players.

Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1

Finite automaton solution:

```
0/0

S0
```

Games with delay for automaton synthesis · GandALF 2013
Example

Alphabet \( \{0, 1\} \) for both players.

Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1

Finite automaton solution:
Example

Alphabet \(\{0, 1\}\) for both players.

Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1

Finite automaton solution:
Example

Alphabet \{0, 1\} for both players.

Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1

Finite automaton solution:
Example

Alphabet \{0, 1\} for both players.

Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1

Finite automaton solution:
Automatic Relations as Specifications

Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1

0101001000 · · ·
0010001010 · · ·
Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1

```
0 1 0 1 0 0 1 0 0 0 ... 
0 0 1 0 0 0 1 0 1 0 ... 
```
Automatic Relations as Specifications

Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Automatic Relations as Specifications

Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Winning condition for output player:

- each input 1 later followed by output 1
- infinitely often output 0
- between two outputs 1 there is an input 1
Theorem (Büchi/Landweber 1969). The synchronous synthesis problem for $\omega$-automatic specifications is solvable. If the specification is realizable, then a finite automaton solution can be constructed.

Proof idea:

• Use game view of problem.
• Reduce the game with
  • simple rules (players play bits in alternation) but a complex winning condition
  • to a game with more complex rules (played on a finite graph) but much simpler winning condition.
• Compute a strategy in the new game and transfer it back to the initial game.
Outline

1 Automaton Synthesis from Specifications
   - Classical Setting
   - Delaying the Output
   - Beyond Finite Automata

2 Synthesis of Lookahead Delegators
Realizing Specifications with Sequential Transducers

\[
\text{specification} \subseteq \{\text{inputs}\} \times \{\text{outputs}\}
\]
Realizing Specifications with Sequential Transducers

\[
\text{specification} \subseteq \{\text{inputs}\} \times \{\text{outputs}\}
\]

Sequential transducer: can output a finite word for each input letter
Games with Delay

Player output can skip moves or play several symbols at once.
Games with Delay

Player output can skip moves or play several symbols at once.

$I \quad a_0$

$J$
Games with Delay

Player output can skip moves or play several symbols at once.

$I \quad a_0$

$J \quad b_0$
Games with Delay

Player output can skip moves or play several symbols at once.

\[ I \quad a_0 \quad a_1 \]

\[ J \quad b_0 \]
Games with Delay

Player output can skip moves or play several symbols at once.

\[
I \quad a_0 \ a_1
\]

\[
J \quad b_0 \text{ skip}
\]
Games with Delay

Player output can skip moves or play several symbols at once.

\[ I \quad a_0 \quad a_1 \quad a_2 \]

\[ J \quad b_0 \]
Games with Delay

Player output can skip moves or play several symbols at once.

\[ I \quad a_0 \quad a_1 \quad a_2 \]

\[ J \quad b_0 \quad b_1 \]
Games with Delay

Player output can skip moves or play several symbols at once.

\[ I \quad a_0 \ a_1 \ a_2 \ a_3 \]

\[ J \quad b_0 \ b_1 \]
Games with Delay

Player output can skip moves or play several symbols at once.

\[ I \quad a_0 \ a_1 \ a_2 \ a_3 \]

\[ J \quad b_0 \ b_1 \ \text{skip} \]
Games with Delay

Player output can skip moves or play several symbols at once.

\[ I \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad a_4 \]

\[ J \quad b_0 \quad b_1 \]
Games with Delay

Player output can skip moves or play several symbols at once.

\[ I \quad a_0 \ a_1 \ a_2 \ a_3 \ a_4 \]

\[ J \quad b_0 \ b_1 \ b_2 \ b_3 \]
Games with Delay

Player output can skip moves or play several symbols at once.

\[ I \quad a_0 \ a_1 \ a_2 \ a_3 \ a_4 \cdots \]

\[ J \quad b_0 \ b_1 \ b_2 \ b_3 \cdots \quad \in \text{specification} \]
Games with Delay

Player output can skip moves or play several symbols at once.

\[ I \quad a_0 \; a_1 \; a_2 \; a_3 \; a_4 \cdots \]
\[ J \quad b_0 \; b_1 \; b_2 \; b_3 \cdots \in \text{specification} \]

A finite automaton winning strategy for II in a delay game corresponds to a sequential transducer realizing the specification.
Examples

$I = J = \{0, 1\}$

- specification = “output at $i$ equals input at $i + 1$”
Examples

\[ I = J = \{0, 1\} \]

- specification = “output at \( i \) equals input at \( i + 1 \)”
  Output has to skip once at the beginning.

\[ I \]

\[ J \]
Examples

\( I = J = \{0, 1\} \)

- specification = “output at \( i \) equals input at \( i + 1 \)”
  Output has to skip once at the beginning.

\[
\begin{array}{c}
  I & 0 \\
  J & \\
\end{array}
\]
Examples

\[ I = J = \{0, 1\} \]

- specification = “output at \( i \) equals input at \( i + 1 \)”
  Output has to skip once at the beginning.

\[
\begin{array}{c|c}
I & 0 \\
J & \text{skip}
\end{array}
\]
Examples

\[ I = J = \{0, 1\} \]

- specification = “output at \( i \) equals input at \( i + 1 \)”
  Output has to skip once at the beginning.

\[
\begin{array}{ll}
I & 0 \quad 0 \\
J & \\
\end{array}
\]
Examples

$I = J = \{0, 1\}$

- specification = “output at $i$ equals input at $i + 1$”
  Output has to skip once at the beginning.

\[
\begin{array}{ccc}
I & 0 & 0 \\
J & 0 &
\end{array}
\]
Examples

$I = J = \{0, 1\}$

- specification = “output at $i$ equals input at $i + 1$”
  Output has to skip once at the beginning.

$I \quad 0 \quad 0 \quad 1 \cdots$

$J \quad 0$
Examples

$I = J = \{0, 1\}$

- specification = “output at $i$ equals input at $i + 1$”
  Output has to skip once at the beginning.

$I \quad 0 \quad 0 \quad 1 \cdots$

$J \quad 0 \quad 1 \cdots$
Examples

$I = J = \{0, 1\}$

- specification = “output at $i$ equals input at $i + 1$”
  Output has to skip once at the beginning.
  
  \[
  I \quad 0 \quad 0 \quad 1 \cdots
  \]

  \[
  J \quad 0 \quad 1 \cdots
  \]

- “start output with 1 iff there is 1 somewhere in the input”
Examples

\[ I = J = \{0, 1\} \]

- specification = “output at \( i \) equals input at \( i + 1 \)”
  
  Output has to skip once at the beginning.
  
  \[
  \begin{array}{ccc}
  I & 0 & 0 & 1 \cdots \\
  J & 0 & 1 \cdots \\
  \end{array}
  \]

- “start output with 1 iff there is 1 somewhere in the input”
  
  There is no strategy with delay for this specification.
  
  \[
  \begin{array}{c}
  I \\
  J \\
  \end{array}
  \]
Examples

\[ I = J = \{0, 1\} \]

- specification = “output at \( i \) equals input at \( i + 1 \)”
  
  Output has to skip once at the beginning.

\[
\begin{array}{cccc}
I & 0 & 0 & 1 \cdots \\
J & 0 & 1 \cdots \\
\end{array}
\]

- “start output with 1 iff there is 1 somewhere in the input”
  
  There is no strategy with delay for this specification.

\[
\begin{array}{c}
I & 0 \\
J \\
\end{array}
\]
Examples

\[ I = J = \{0, 1\} \]

- specification = “output at \( i \) equals input at \( i + 1 \)”
  Output has to skip once at the beginning.
  \[
  \begin{array}{cccc}
  I & 0 & 0 & 1 \cdots \\
  \end{array}
  \]
  \[
  \begin{array}{c}
  J \quad 0 \\
  \end{array}
  \]

- “start output with 1 iff there is 1 somewhere in the input”
  There is no strategy with delay for this specification.
  \[
  \begin{array}{c}
  I \quad 0 \\
  \end{array}
  \]
  \[
  \begin{array}{c}
  J \quad \text{skip} \\
  \end{array}
  \]
Examples

$I = J = \{0, 1\}$

- specification = “output at $i$ equals input at $i + 1$”
  Output has to skip once at the beginning.
  \[
  \begin{array}{cccc}
  I & 0 & 0 & 1 \cdots \\
  J & 0 & 1 \cdots \\
  \end{array}
  \]

- “start output with 1 iff there is 1 somewhere in the input”
  There is no strategy with delay for this specification.
  \[
  \begin{array}{cc}
  I & 0 \ 0 \\
  J & \\
  \end{array}
  \]
Examples

$I = J = \{0, 1\}$

- specification = “output at $i$ equals input at $i + 1$”
  Output has to skip once at the beginning.
  
  $I$  
  
  0 0 1 · · ·  

  $J$  
  
  0 1 · · ·  

- “start output with 1 iff there is 1 somewhere in the input”
  
  There is no strategy with delay for this specification.
  
  $I$  
  
  0 0  

  $J$  
  
  skip
Examples

\[ I = J = \{0, 1\} \]

- specification = “output at \( i \) equals input at \( i + 1 \)”
  Output has to skip once at the beginning.
  \[ I \quad 0 \quad 0 \quad 1 \cdots \]
  \[ J \quad 0 \quad 1 \cdots \]

- “start output with 1 iff there is 1 somewhere in the input”
  There is no strategy with delay for this specification.
  \[ I \quad 0 \quad 0 \quad 0 \]
  \[ J \]
Examples

\[ I = J = \{0, 1\} \]

- specification = “output at \( i \) equals input at \( i + 1 \)”
  Output has to skip once at the beginning.
  
  \[
  \begin{array}{cccc}
  I & 0 & 0 & 1 \cdots \\
  J & 0 & 1 \cdots \\
  \end{array}
  \]

- “start output with 1 iff there is 1 somewhere in the input”
  There is no strategy with delay for this specification.
  
  \[
  \begin{array}{ccc}
  I & 0 & 0 \\
  J & \text{skip} \\
  \end{array}
  \]
Examples

\[ I = J = \{0, 1\} \]

- specification = “output at \( i \) equals input at \( i + 1 \)”
  Output has to skip once at the beginning.
  \[
  I \quad 0 \quad 0 \quad 1 \cdots \\
  J \quad 0 \quad 1 \cdots 
  \]

- “start output with 1 iff there is 1 somewhere in the input”
  There is no strategy with delay for this specification.
  \[
  I \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \\
  J 
  \]
Examples

$I = J = \{0, 1\}$

- specification = “output at $i$ equals input at $i + 1$”
  
  Output has to skip once at the beginning.
  
  \[
  \begin{array}{lccc}
  I & 0 & 0 & 1 \cdots \\
  J & 0 & 1 \cdots \\
  \end{array}
  \]

- “start output with 1 iff there is 1 somewhere in the input”

  There is no strategy with delay for this specification.
  
  \[
  \begin{array}{lccc}
  I & 0 & 0 & 0 & 0 \\
  J & 0 \\
  \end{array}
  \]
Examples

\[ I = J = \{0, 1\} \]

- specification = “output at \( i \) equals input at \( i + 1 \)”
  Output has to skip once at the beginning.
  \[
  \begin{array}{cccc}
  I & 0 & 0 & 1 \cdots \\
  J & 0 & 1 \cdots \\
  \end{array}
  \]

- “start output with 1 iff there is 1 somewhere in the input”
  There is no strategy with delay for this specification.
  \[
  \begin{array}{ccccccc}
  I & 0 & 0 & 0 & 0 & 0 & 1 \cdots \\
  J & 0 \\
  \end{array}
  \]
Bounded Delay in $\omega$-Regular Games

Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). For $\omega$-automatic specifications it is decidable if there is a strategy with delay realizing the specification. Furthermore, strategies with bounded delay are sufficient.
Bounded Delay in $\omega$-Regular Games

Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). For $\omega$-automatic specifications it is decidable if there is a strategy with delay realizing the specification. Furthermore, strategies with bounded delay are sufficient.

Corollary. It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer.
Bounded Delay in $\omega$-Regular Games

Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). For $\omega$-automatic specifications it is decidable if there is a strategy with delay realizing the specification. Furthermore, strategies with bounded delay are sufficient.

Corollary. It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer.

Bounded delay is sufficient basically because Player output has to produce an infinite sequence for each input.

$\leadsto$ What about finite words?
Finite Words

Given: Specification as automatic relation over finite words.

Question: Does there exist a sequential transducer implementing the specification?
Example: Alphabet \( \{a, b, c\} \)

\[
R = (ac^*b, bc^*a) \cup (bc^*a, ac^*b) \cup (ac^*a, ac^*a) \cup (bc^*b, bc^*b)
\]
Example: Alphabet \( \{a, b, c\} \)

\[
R = (ac^*b, bc^*a) \cup (bc^*a, ac^*b) \cup (ac^*a, ac^*a) \cup (bc^*b, bc^*b)
\]
Example: Alphabet \{a, b, c\}

\[ R = (ac^*b, bc^*a) \cup (bc^*a, ac^*b) \cup (ac^*a, ac^*a) \cup (bc^*b, bc^*b) \]
Automatic Relations over Finite Words

Example: Alphabet \( \{a, b, c\} \)

\[
R = (ac^* b, bc^* a) \cup (bc^* a, ac^* b) \cup (ac^* a, ac^* a) \cup (bc^* b, bc^* b)
\]
Automatic Relations over Finite Words

Example: Alphabet \( \{a, b, c\} \)

\[
R = (ac^*b, bc^*a) \cup (bc^*a, ac^*b) \cup (ac^*a, ac^*a) \cup (bc^*b, bc^*b)
\]
Example: Alphabet \(\{a, b, c\}\)

\[
R = (ac^* b, bc^* a) \cup (bc^* a, ac^* b) \cup (ac^* a, ac^* a) \cup (bc^* b, bc^* b)
\]
Example: Alphabet \( \{a, b, c\} \)

\[
R = (ac^*b, bc^*a) \cup (bc^*a, ac^*b) \cup (ac^*a, ac^*a) \cup (bc^*b, bc^*b)
\]
Automatic Relations over Finite Words

Example: Alphabet \( \{a, b, c\} \)

\[
R = (ac^*b, bc^*a) \cup (bc^*a, ac^*b) \cup (ac^*a, ac^*a) \cup (bc^*b, bc^*b)
\]
Example: Unbounded Delay

\[ R = (ac^*b, bc^*a) \cup (bc^*a, ac^*b) \cup (ac^*a, ac^*a) \cup (bc^*b, bc^*b) \]

Realization by sequential transducer:
Result

Theorem (Carayol/L.’12). For an automatic specification (over finite words) it is decidable whether it can be realized by a sequential transducer.

Proof idea:

- Either the delay remains within a computable bound $K$ (use the standard game theory techniques),
- or the sequential transducer can delay its output until the whole input is known.
Outline

1 Automaton Synthesis from Specifications
   - Classical Setting
   - Delaying the Output
   - Beyond Finite Automata

2 Synthesis of Lookahead Delegators
Beyond Finite Automata

Use pushdown automata (finite automata + stack) instead of finite automata.

Without Delay:

Theorem (Walukiewicz’96). The synchronous synthesis problem for deterministic pushdown specifications is decidable. If the specification is realizable, then it can be implemented by a pushdown automaton.
Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

$$(0)^\omega$$ or $$(0)$$
Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

$$
\begin{align*}
(0) \omega \\
(0) \omega \\
(0^n 0^n 1^n I) \omega \\
(0^n 0^n 1^n J) \omega
\end{align*}
$$
Beyond Finite Automata

With Delay:

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

\[
\begin{align*}
(0)^\omega & \quad \text{or} \quad (0)^n (0)^n (1)_I (I)_J^\omega \\
(0) & \quad \text{or} \quad (0)^n (0)^{n+1} (1)_I (I)_J^\omega
\end{align*}
\]
Beyond Finite Automata

With Delay:

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

\[
\begin{align*}
(0)^\omega \text{ or } (0)^n (0)^n (1) (I)^\omega \\
I^n (0)^n (1) (J)^\omega \text{ or } (0)^n (0)^n+1 (1) (I) (J)^\omega
\end{align*}
\]

There is a strategy with linear delay:

$I$

$J$
Beyond Finite Automata

With Delay:

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

\[
\begin{array}{c}
(0)^\omega \\
0 \\
0
\end{array}
\quad \text{or} \quad
\begin{array}{c}
0^n (0)^n (1) (I)^\omega \\
0^n (1) (I) (I)
\end{array}
\quad \text{or} \quad
\begin{array}{c}
0^n (0)^{n+1} (1) (I) (I)^\omega \\
0^n (1) (I) (I)
\end{array}
\]

There is a strategy with linear delay:

\[
\begin{array}{c}
I \\
0 \\
J
\end{array}
\]
Beyond Finite Automata

With Delay:

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

$$
\begin{align*}
(0)^\omega & \text{ or } (0)^n (0)^n (1) (I)^\omega \\
& \text{ or } (0)^n (0)^{n+1} (1) (I) (I)^\omega
\end{align*}
$$

There is a strategy with linear delay:

$I$ 0

$J$ skip
Beyond Finite Automata

With Delay:

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

\[
\begin{align*}
(0)^\omega & \text{ or } (0)^n (0)^n (1) (I)^\omega \text{ or } (0)^n (0)^{n+1} (1) (I)^\omega \\
0 & \text{ or } 0^n (1) (I) (I) \\
0 & \text{ or } 0^n (1) (1) (I) (I)
\end{align*}
\]

There is a strategy with linear delay:

$I$ \quad 0 \quad 0$

$J$
Example: Specification allows the following pairs of input/output sequences (with \( I = J = \{0, 1\} \)):

\[
\begin{align*}
(0)^\omega & \text{ or } (0)^n (0)^n (1) (I)^\omega \\
(0) & \text{ or } (0)^n (0)^{n+1} (1) (I) (J)
\end{align*}
\]

There is a strategy with linear delay:

\[
\begin{align*}
I & \quad 0 \quad 0 \\
J & \quad 0
\end{align*}
\]
Beyond Finite Automata

With Delay:

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

$$(0)^\omega \quad \text{or} \quad (0)^n (0)^n (1) (I)^\omega \quad \text{or} \quad (0)^n (0)^{n+1} (1) (I)^\omega$$

There is a strategy with linear delay:

$I \quad 0 \quad 0 \quad 0$

$J \quad 0$
Beyond Finite Automata

With Delay:

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

\[
\begin{align*}
&\left(\begin{array}{c}
0 \\
0
\end{array}\right) \omega \\
&\text{or} \\
&\left(\begin{array}{c}
0 \\
0
\end{array}\right)^n \left(\begin{array}{c}
0 \\
1
\end{array}\right)^n \left(\begin{array}{c}
I \\
J
\end{array}\right) \omega \\
&\text{or} \\
&\left(\begin{array}{c}
0 \\
0
\end{array}\right)^n \left(\begin{array}{c}
0 \\
1
\end{array}\right)^{n+1} \left(\begin{array}{c}
1 \\
J
\end{array}\right) \left(\begin{array}{c}
I \\
J
\end{array}\right) \omega
\end{align*}
\]

There is a strategy with linear delay:

\[
\begin{align*}
I & \quad 0 \quad 0 \quad 0 \\
J & \quad 0 \quad \text{skip}
\end{align*}
\]
Example: Specification allows the following pairs of input/output sequences (with \( I = J = \{0, 1\} \)):

\[
\begin{align*}
(0)^\omega \quad \text{or} \quad (0)^n (0)^n (1)^\omega \quad \text{or} \quad (0)^n (0)^{n+1} (1) (I)^\omega
\end{align*}
\]

There is a strategy with linear delay:

\[
\begin{align*}
I & \quad 0 \quad 0 \quad 0 \quad 0 \\
J & \quad 0
\end{align*}
\]
Beyond Finite Automata

With Delay:

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

$$
\begin{align*}
(0) \omega & \text{ or } (0)^n (0)^n (1) (I)^\omega & \text{ or } (0)^n (0)^{n+1} (1) (I)^\omega
\end{align*}
$$

There is a strategy with linear delay:

$I$

\begin{tabular}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
\end{tabular}

$J$

\begin{tabular}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
\end{tabular}
Beyond Finite Automata

With Delay:

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

\[
\begin{align*}
\left( \begin{array}{c} 0 \\ 0 \end{array} \right)^\omega & \text{ or } \left( \begin{array}{c} 0 \\ 0 \end{array} \right)^n \left( \begin{array}{c} 0 \\ 1 \end{array} \right)^n \left( \begin{array}{c} 1 \\ I \end{array} \right)^\omega & \text{ or } \left( \begin{array}{c} 0 \\ 0 \end{array} \right)^n \left( \begin{array}{c} 0 \\ 1 \end{array} \right)^{n+1} \left( \begin{array}{c} 1 \\ J \end{array} \right)^\omega
\end{align*}
\]

There is a strategy with linear delay:

\[
\begin{array}{c}
I \\
J
\end{array}
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]
Beyond Finite Automata

With Delay:

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

$$
\begin{align*}
(0) \omega & \quad \text{or} \quad (0)^n (0) (1) (I) \omega \\
(0) (0) & \quad \text{or} \quad (0)^n (0) (1)^n (1) (I) \omega \\
(1) & \quad \text{or} \quad (1)^n (1) (I) \omega
\end{align*}
$$

There is a strategy with linear delay:

$I$  
0 0 0 0 0 0

$J$  
0 0 skip
Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

\[
\left(\begin{array}{c}
0 \\
0
\end{array}\right)^{\omega} \quad \text{or} \quad \left(\begin{array}{c}
0 \\
0
\end{array}\right)^n \left(\begin{array}{c}
0 \\
1
\end{array}\right)^n \left(\begin{array}{c}
1 \\
I
\end{array}\right) \left(\begin{array}{c}
I \\
J
\end{array}\right)^{\omega} \quad \text{or} \quad \left(\begin{array}{c}
0 \\
0
\end{array}\right)^n \left(\begin{array}{c}
0 \\
1
\end{array}\right)^{n+1} \left(\begin{array}{c}
1 \\
J
\end{array}\right) \left(\begin{array}{c}
I \\
J
\end{array}\right)^{\omega}
\]

There is a strategy with linear delay:

\[
I \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \\
J \quad 0 \quad 0
\]
Example: Specification allows the following pairs of input/output sequences (with $I = J = \{0, 1\}$):

\[
\begin{align*}
(0)^\omega & \quad \text{or} \quad (0^n 0^n 1^n I^n J^n)^\omega \quad \text{or} \quad (0^n 0^n 1^n 1^n I^n J^n)^\omega
\end{align*}
\]

There is a strategy with linear delay:

\begin{align*}
I & \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \ldots \\
J & \quad 0 \quad 0 \quad 1 \ldots
\end{align*}
Beyond Finite Automata

Theorem (Fridman/L./Zimmerman’11).

- There are deterministic pushdown specifications for which there is a delay strategy but each such strategy needs non-elementary delay.
- For deterministic pushdown specifications it is undecidable if there is a strategy with delay realizing the specification.
Outline

1. Automaton Synthesis from Specifications
   - Classical Setting
   - Delaying the Output
   - Beyond Finite Automata

2. Synthesis of Lookahead Delegators
**Problem Setting**

Given an NFA, decide whether it can deterministically choose its transitions using a bounded lookahead.

A function choosing the transitions with lookahead $k$ is called a $k$-lookahead delegator for $\mathcal{A}$.

**Decision Problem:** Given NFA $\mathcal{A}$ and a number $k$, does there exist a $k$-lookahead delegator for $\mathcal{A}$?
Problem Setting

Given an NFA, decide whether it can deterministically choose its transitions using a bounded lookahead.

A function choosing the transitions with lookahead \( k \) is called a \( k \)-lookahead delegator for \( A \).

Decision Problem: Given NFA \( A \) and a number \( k \), does there exist a \( k \)-lookahead delegator for \( A \)?

In this example the two symbols after the current input letter are needed to choose a transition.

\( \sim \) 2-lookahead delegator
The NFA guesses the last letter.

To choose the first transition, the last input letter needs to be known.
Game Approach

Game for $\mathcal{A}$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $\mathcal{A}$.

![Automaton Diagram]

Games with delay for automaton synthesis · GandALF 2013
Game Approach

Game for $A$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $A$.
Game Approach

Game for $\mathcal{A}$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $\mathcal{A}$.

**Diagram**

- Input word: $aaa$.
- Run: $q_{aa} \rightarrow q_{ba} \rightarrow q_{bb}$.
- Skip: $q_{aa}$.

**Graph**

- States: $q_{aa}$, $q_{ba}$, $q_{bb}$, $q_{ab}$.
- Edges: $a \rightarrow q_{aa}$, $b \rightarrow q_{ba}$, $a \rightarrow q_{ab}$, $b \rightarrow q_{bb}$, $a \rightarrow q_{aa}$, $b \rightarrow q_{bb}$.
Game Approach

Game for $A$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $A$.

```
    q_{aa}     q_{ba}
       |            |
       |            |
       a           b

    q_{ab}     q_{bb}
       |            |
       |            |
       a           b

input  a    a
run    q_{aa}
```
Game Approach

Game for $A$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $A$.
Game Approach

Game for $A$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $A$.

![Automaton Diagram]

Input: $a a a a$

Run: $q_{aa}$
Game Approach

Game for $A$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $A$.

```
input  a  a  a  a
run    qaa  qaa
```

```
\includegraphics[width=0.4\textwidth]{game_diagram}
```
Game Approach

Game for $\mathcal{A}$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $\mathcal{A}$.

```
input   a   a   a   a   b
run     q_{aa} q_{aa}
```

---

Games with delay for automaton synthesis · GandALF 2013
Game Approach

Game for $\mathcal{A}$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $\mathcal{A}$.

![Automaton Diagram]

Input: $a\ a\ a\ a\ b$

Run: $q_{aa}\ q_{aa}\ q_{ab}$
Game Approach

Game for $A$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $A$.

```
input a a a a b
run qaa qaa qab
```
Game Approach

Game for $\mathcal{A}$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $\mathcal{A}$.

\[
\begin{aligned}
 q_{aa} \xrightarrow{a} q_{ba} \xleftarrow{b} q_{ab} \xrightarrow{a} q_{bb} \\
 q_{ab} \xrightarrow{a} q_{ba} \xrightarrow{b} q_{bb} \\
 q_{ab} \xrightarrow{b} q_{bb} \xrightarrow{b} q_{ba}
\end{aligned}
\]

input \hspace{1cm} a \hspace{1cm} a \hspace{1cm} a \hspace{1cm} a \hspace{1cm} b \hspace{1cm} \triangleleft

run \hspace{1cm} q_{aa} \hspace{1cm} q_{aa} \hspace{1cm} q_{ab} \hspace{1cm} q_{ba}
Game Approach

Game for $A$ and $k$:

- Player I (input) plays an input word.
- Player II (run) plays a run on the input word, and can delay its choices up to $k$ letters.
- In order to win, II has to construct an accepting run if the input is accepted by $A$.
Problem with this Approach

- The winning condition requires to check whether the input is accepted by $A$.
- This makes the game difficult to solve.

Solution

- Player input also chooses a transition, but after Player run has chosen one.
- New winning condition: if player input ends in a final state, then player run also has to end in a final state.
Extended Game

Games with delay for automaton synthesis · GandALF 2013
Extended Game

- Input: $q_{aa} \ a$
- Run: $q_{aa}$

Diagram:

- States: $q_{aa}, q_{ba}, q_{ab}, q_{bb}$
- Transitions:
  - $q_{aa} \xrightarrow{a} q_{aa}$
  - $q_{aa} \xrightarrow{b} q_{ba}$
  - $q_{ba} \xrightarrow{a} q_{ab}$
  - $q_{ba} \xrightarrow{b} q_{bb}$
  - $q_{ab} \xrightarrow{a} q_{aa}$
  - $q_{ab} \xrightarrow{b} q_{bb}$
  - $q_{bb} \xleftarrow{a}$
  - $q_{bb} \xleftarrow{b}$
Extended Game

input \( q_{aa} \ a \)

run \( q_{aa} \) skip
Extended Game

input $q_{aa} \ a \ a$

run $q_{aa}$
input $q_{aa} \ a \ a$

run $q_{aa} \ q_{aa}$
Extended Game

input $q_{aa}$ $a$ $q_{ab}$ $a$ $b$

run $q_{aa}$ $q_{aa}$
input \( q_{aa} \ a \ q_{ab} \ a \ b \)

run \( q_{aa} \ q_{aa} \ q_{ab} \)
Extended Game

input \( qa a \ q ab \ a \ q ba \ b \)

run \( qa a \ q aa \ q ab \)

Games with delay for automaton synthesis · GandALF 2013
input $\mathit{q_{aa}} \ a \ \mathit{q_{ab}} \ a \ \mathit{q_{ba}} \ b \ \mathit{q_{aa}} \triangleleft$

run $\mathit{q_{aa}} \ \mathit{q_{aa}} \ \mathit{q_{ab}}$
This extended game can be viewed as a safety game of size roughly $|Q|^2 \times |\Sigma|^{k+1}$.

A winning strategy for II corresponds to a $k$-lookahead delegator for $A$. 

\[
\begin{align*}
\text{input} & \quad q_{aa} & a & q_{ab} & a & q_{ba} & b & q_{aa} \\
\text{run} & \quad q_{aa} & q_{aa} & q_{ab}
\end{align*}
\]
Result

Theorem (L./Repke’13). Let $k$ be a fixed number. It is decidable in polynomial time whether a given NFA $\mathcal{A}$ has a $k$-lookahead delegator.

Remark: For $k = 0$ this corresponds to deciding whether $\mathcal{A}$ has an equivalent deterministic subautomaton.
Conclusion

Games with delay as a useful tool for synthesis problems in automata theory:

- Sequential transducers from automatic specifications
- Lookahead delegators for NFAs

Some open problems:

- Synthesis of tree transducers
- Lookahead delegators for $\omega$-automata