Synthesis of Transducers from Automatic Specifications

Christof Löding
RWTH Aachen University, Germany

Highlights of Logic, Games and Automata
Paris
September 3, 2014
Given a (binary) relation $R$ (in this talk over words or trees), synthesize a function $f_R$

- that has the same domain as $R$, and
- whose graph is a subset of $R$.

We say that $f_R$ implements (or realizes) the specification $R$. 

**Diagram:**

- **Yellow Box:** $R$
  - class of specifications
- **Blue Box:** $f_R$
  - class of implementations

**Arrow:** Synthesize $f_R$ from $R$.
Specifications

Alphabets $I$ and $J$ for input and output

Finite automata for word relations: use two tapes

- **Asynchronous automata**: transitions are labeled with pairs of words $x/y$ or $\left(\frac{x}{y}\right)$ with $x \in I^*$ and $y \in J^*$.
  $\leadsto$ rational relations

- **Synchronous automata**: transitions are labeled with pairs of letters $a/b$ or $\left(\frac{a}{b}\right)$ with $a \in I$ and $b \in J$ (using a padding symbol in case of finite words).
  $\leadsto$ automatic relations

Acceptance: standard acceptance conditions for finite/infinite words
Examples for Finite Words

An automatic relation: \( R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \)

1 1 1 1 1 1 1 1 □ □ □
1 1 1 1 1 0 1 1 0 0 0
Examples for Finite Words

An automatic relation: \( R = \{ (1^n, 1^m01^k0^*) \mid n = m + k + 1 \} \)
Examples for Finite Words

An automatic relation: \[ R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \]
Examples for Finite Words

An automatic relation: \( R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \)
Examples for Finite Words

An automatic relation: \( R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \)
Examples for Finite Words

An automatic relation: \( R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \)
Examples for Finite Words

An automatic relation: \( R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \)
Examples for Finite Words

An automatic relation: \( R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \)
Examples for Finite Words

An automatic relation: $R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\}$
Examples for Finite Words

An automatic relation: \( R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \)
Examples for Finite Words

An automatic relation: \( R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \)
Examples for Finite Words

An automatic relation: \( R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \)
Examples for Finite Words

An automatic relation: \( R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \)
Examples for Finite Words

An automatic relation: \( R = \{(1^n, 1^m01^k0^*) \mid n = m + k + 1\} \)

A rational relation: \( R = \{(v, w) \mid v \text{ is a non-empty infix of } w\} \)
A **sequential transducer** (asynchronous)

- reads one input letter in each step
- is deterministic on the input
- produces an output word (possibly empty) in each step
Implementations: Deterministic Sequential Transducers

A **sequential transducer** (asynchronous)

- reads one input letter in each step
- is deterministic on the input
- produces an output word (possibly empty) in each step

A **synchronous sequential transducer** produces one output letter in each step.

Applies to finite and infinite words.
General Problem Setting

$R$  

class of specifications  

class of specifications  

$\mathcal{R}$  

class of implementations  

class of implementations  

sync/async automata  

transducers
Outline

1. Classical Setting – Büchi-Landweber Theorem
2. Transducer Synthesis for Finite Words
3. Trees
4. Beyond Finite Automata
Motivation: Realizing Specifications

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

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

synthesize

input $\rightarrow$ program $P$ $\rightarrow$ output
Motivation: Realizing Specifications

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

synthesize

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

input

output

Synthesis of Transducers from Automatic Specifications · Highlights'14
Motivation: Realizing Specifications

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

synthesize

program $P$

input $a_0$ 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 \rightarrow \quad b_0 \leftarrow \quad P
Motivation: Realizing Specifications

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

synthesize

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

\[ \text{input} \]
\[ a_0 \quad a_1 \]

\[ \text{output} \]
\[ b_0 \]
Motivation: Realizing Specifications

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

synthesize

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

input $a_0 \ a_1$

output $b_0 \ b_1$
Motivation: Realizing Specifications

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

synthesize

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

input: \ a_0 \ a_1 \ a_2

output: \ b_0 \ b_1
Motivation: Realizing Specifications

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

synthesize

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

input

\[ a_0 \ a_1 \ a_2 \]

output

\[ b_0 \ b_1 \ b_2 \]
Motivation: Realizing Specifications

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

synthesize

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

input $\quad a_0 \ a_1 \ a_2 \cdots$

output $\quad b_0 \ b_1 \ b_2 \cdots \in \text{specification}$
Viewed as a Game

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

```
I  a_0
II
```
Viewed as a Game

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

\[
\begin{array}{c|c}
I & a_0 \\
II & b_0 \\
\end{array}
\]
Viewed as a Game

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

\[
\begin{array}{c}
| & a_0 & a_1 \\
| & b_0 \\
\end{array}
\]
Two players I (input) and II (output) play letters from finite alphabets ($I$ and $J$) in alternation:

<table>
<thead>
<tr>
<th>I</th>
<th>$a_0$</th>
<th>$a_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>II</td>
<td>$b_0$</td>
<td>$b_1$</td>
</tr>
</tbody>
</table>
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:

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

II
\[ b_0 \quad b_1 \quad b_2 \]
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 \quad a_1 \quad a_2 \ldots

II \quad b_0 \quad b_1 \quad b_2
Viewed as a Game

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

I \hspace{1.5cm} a_0 \ a_1 \ a_2 \cdots

II \hspace{1.5cm} b_0 \ b_1 \ b_2 \cdots
Viewed as a Game

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

I \hspace{1cm} a_0 \; a_1 \; a_2 \cdots

II \hspace{1cm} 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.
Viewed as a Game

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

I  \[ a_0 \ a_1 \ a_2 \cdots \]

II \[ b_0 \ b_1 \ b_2 \cdots \]

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 synchronous sequential transducer
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:
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:

\[ S_0 \xrightarrow{0/0} S_0 \xrightarrow{1/0} S_1 \]
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:

![Finite automaton](image-url)
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:
Büchi-Landweber Theorem

Theorem (Büchi/Landweber 1969). Given an $\omega$-automatic specification, it is decidable whether this specification can be realized by a synchronous transducer.

Proof idea:

- Use game view of problem.
- Build a finite game arena by running (a determinization of the) specification automaton on the letters chosen by the players.

\[
q \xrightarrow{I} (q, a) \xrightarrow{II} (q, a, b) \rightarrow \delta(q, a/b)
\]

$\text{II}$ wants to make the automaton accept.

- A winning strategy with finite memory for $\text{II}$ corresponds to a synchronous transducer.
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

- Player output can skip moves or play several symbols at once.
  \[
  I \quad a_0 \quad J
  \]
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an \(\omega\)-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

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

\[
\begin{array}{c|c}
I & a_0 \\
J & b_0 \\
\end{array}
\]
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

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

$I \quad a_0 \ a_1$

$J \quad b_0$
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

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

\[
\begin{array}{c}
I & a_0 & a_1 \\
J & b_0 & \text{skip}
\end{array}
\]
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

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

\[
\begin{array}{c}
I & a_0 & a_1 & a_2 \\
J & b_0 \\
\end{array}
\]
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

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

$I \quad a_0 \ a_1 \ a_2$

$J \quad b_0 \ b_1$
Sythesizing Sequential Transducers

Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

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

```
I    a_0 a_1 a_2 a_3
J    b_0 b_1
```
Sythesizing Sequential Transducers

Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

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

$I \quad a_0 \ a_1 \ a_2 \ a_3$

$J \quad b_0 \ b_1 \ \text{skip}$
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

- 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
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

- Player output can skip moves or play several symbols at once.
  
  \[
  I \quad a_0 \ a_1 \ a_2 \ a_3 \ a_4 \\
  J \quad b_0 \ b_1 \ b_2 \ b_3
  \]
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

- Player output can skip moves or play several symbols at once.
  
  \[
  \begin{array}{c}
  I \\
  a_0 \ a_1 \ a_2 \ a_3 \ a_4 \cdots \\
  \hline
  J \\
  b_0 \ b_1 \ b_2 \ b_3 \cdots \\
  \end{array}
  \]

  $\in$ specification
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

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

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

- A finite memory winning strategy for II in the asynchronous game corresponds to a sequential transducer realizing the specification.
Theorem (Hosch/Landweber’72, Holtmann/Kaiser/Thomas’10). It is decidable whether an $\omega$-automatic specification can be realized by a sequential transducer. Furthermore, sequential transducers with bounded delay are sufficient.

Proof idea: Use an asynchronous game to model the problem.

- Player output can skip moves or play several symbols at once.
  $$ I \quad a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ldots $$
  $$ J \quad b_0 \ b_1 \ b_2 \ b_3 \ldots \in \text{specification} $$

- A finite memory winning strategy for II in the asynchronous game corresponds to a sequential transducer realizing the specification.

- Bounded delay is sufficient because Player output has to produce an infinite sequence for each input. $\leadsto$ finite game arena.
Outline

1. Classical Setting – Büchi-Landweber Theorem
2. Transducer Synthesis for Finite Words
3. Trees
4. Beyond Finite Automata
Decision Problems

1. **asynchronous**: Given a rational relation, can it be realized by a sequential transducer?

2. **synchronous**: Given an automatic relation, can it be realized by a synchronous sequential transducer?
   (decidable: Büchi-Landweber theorem for finite words)
Decision Problems

1. asynchronous: Given a rational relation, can it be realized by a sequential transducer?
2. synchronous: Given an automatic relation, can it be realized by a synchronous sequential transducer? (decidable: Büchi-Landweber theorem for finite words)
3. async/sync: Given a rational relation, does it have a uniformization by a synchronous sequential transducer?
4. sync/async: Given an automatic relation, does it have a uniformization by a sequential transducer?
Theorem. (Carayol/L.) The following problem is undecidable: Given a rational relation, can it be realized by a sequential transducer?
Theorem. (Carayol/L.) The following problem is undecidable: Given a rational relation, can it be realized by a sequential transducer?

Idea: Reduction from halting problem for Turing machines.

Specification

Input: \( \#c_0\#c_1 \cdots \#c_n\#^* A/B \)
Output: \( \#c'_0\#c'_1 \cdots \#c'_n\#^* A/B \)

- \( c_0 \) initial, \( c_n \) halting configuration, \( |input| = |output| \)
Undecidability for Rational Relations

Theorem. (Carayol/L.) The following problem is undecidable: Given a rational relation, can it be realized by a sequential transducer?

Idea: Reduction from halting problem for Turing machines.

Specification

input: \#c_0\#c_1 \cdots \#c_n\#* A / B
output: \#c'_0\#c'_1 \cdots \#c'_n\#* A / B

- $c_0$ initial, $c_n$ halting configuration, $|\text{input}| = |\text{output}|$
- If last letter of input is $A$, then input $=$ output.
- If last letter of input is $B$, then for some $i$, configuration $c'_{i+1}$ is not successor configuration of $c_i$. 

Synthesis of Transducers from Automatic Specifications · Highlights’14
Undecidability for Rational Relations

Theorem. (Carayol/L.) The following problem is undecidable: Given a rational relation, can it be realized by a sequential transducer?

Idea: Reduction from halting problem for Turing machines.

Specification

input: \#c_0\#c_1 \cdots \#c_n\#^* A/B
output: \#c'_0\#c'_1 \cdots \#c'_n\#^* A/B

- $c_0$ initial, $c_n$ halting configuration, $|\text{input}| = |\text{output}|$
- If last letter of input is $A$, then input = output.
- If last letter of input is $B$, then for some $i$, configuration $c'_{i+1}$ is not successor configuration of $c_i$.

Implementation:

- TM does not halt: simply copy the input word
- TM halts: if $c_0 \cdots c_n$ is the halting computation, then it is necessary to know the last input letter to produce a valid output sequence. $\leadsto$ no finite state transducer
Decision Problems

1. asynchronous: Given a rational relation, can it be realized by a sequential transducer? *(undecidable)*

2. synchronous: Given an automatic relation, can it be realized by a synchronous sequential transducer? *(decidable: Büchi-Landweber theorem for finite words)*

3. async/sync: Given a rational relation, can it be realized by a synchronous sequential transducer? *(undecidable)*

4. sync/async: Given an automatic relation, can it be realized by a sequential transducer?
Example: Unbounded Delay

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

\[ a \ c \ c \ c \ c \ b \]
\[ b \ c \ c \ a \quad \text{“swap first and last letter”} \]

Realization by sequential transducer: \[ xc^*y \mapsto yx \]
Result

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

Proof idea:

• Use the asynchronous game.
• To model finite words, player input has to play an endmarker eventually. Then player output makes the last move.
• Problem: the delay cannot be bounded
• Solution: if the delay exceeds a certain bound $K$, then output can wait until the end, and play a last move based on some regular information of the input.
Decision Problems

1. asynchronous: Given a rational relation, can it be realized by a sequential transducer? (undecidable)

2. synchronous: Given an automatic relation, can it be realized by a synchronous sequential transducer? (decidable: Büchi-Landweber theorem for finite words)

3. async/sync: Given a rational relation, can it be realized by a synchronous sequential transducer? (undecidable)

4. sync/async: Given an automatic relation, can it be realized by a sequential transducer? (decidable)
Outline

1. Classical Setting – Büchi-Landweber Theorem
2. Transducer Synthesis for Finite Words
3. Trees
4. Beyond Finite Automata
Problem Setting for Trees

- Class of specifications
- Class of implementations

$R$  
synthesize  
$f_R$

Automata for tree relations  
Tree transducers
Relations of Finite Trees

How to combine trees?

finite tree automata on such combined trees $\leadsto$ automatic relations over trees

**Note:** determinism $\neq$ nondeterminism for top-down automata
Implementations: Top-Down Tree Transducer

Transitions are of the following form (example):

\[ q \xrightarrow{f} x_1 \quad x_2 \quad x_3 \quad g \xrightarrow{h} q_1 \quad g \xrightarrow{a} q_2 \quad q_3 \quad b \]
Example

Transitions:

\[
q_0(q(x_1)) \rightarrow f(q(x_1), q(x_1)) \\
q(g(x_1)) \rightarrow h(q((x_1)) \\
q(c) \rightarrow b
\]

Execution:
Example

Transitions:

\[
\begin{align*}
q_0(q(x_1)) & \rightarrow f(q(x_1), q(x_1)) \\
q(g(x_1)) & \rightarrow h(q((x_1)) \\
q(c) & \rightarrow b
\end{align*}
\]

Execution:
Example

Transitions:

\[
\begin{align*}
q_0(q(x_1)) & \rightarrow f(q(x_1), q(x_1)) \\
q(g(x_1)) & \rightarrow h(q((x_1)) \\
q(c) & \rightarrow b
\end{align*}
\]

Execution:
Example

Transitions:

\[ q_0(q(x_1)) \rightarrow f(q(x_1), q(x_1)) \]
\[ q(g(x_1)) \rightarrow h(q((x_1)) \]
\[ q(c) \rightarrow b \]

Execution:
Example

Transitions:

\[ q_0(q(x_1)) \rightarrow f(q(x_1), q(x_1)) \]
\[ q(g(x_1)) \rightarrow h(q((x_1)) \]
\[ q(c) \rightarrow b \]

Execution:

\[
\begin{array}{c}
g \\
g \\
g \\
c \end{array} \quad \begin{array}{c}
f \\
\quad \begin{array}{c}
h \\\n\quad \begin{array}{c}
b \\\n\quad \begin{array}{c}
h \\
\end{array} \end{array} \\
\quad \begin{array}{c}
h \\\n\end{array} \end{array} \\
\end{array}
\]
Deterministic Specifications

**Theorem (L./Winter’14).** It is decidable, given an automatic tree relation defined by a deterministic top-down tree automaton, whether it can be implemented by a top-down tree transducer.

**Idea:**

- If a TDTT exists, there is one that does not swap or copy subtrees.
- Extend game methods from word case.

**Note:** We assume that the transducer needs not to verify the domain of the relation. If an input tree does not have an image under the relation, then the transducer can produce arbitrary output.
Including Domain Verification

Relation:

\[ b \xrightarrow{f} t_a \times t \xrightarrow{f} b \]

\[ t_a : \text{ only } a \text{ at leafs} \]
\[ t : \text{ arbitrary tree} \]

Implementation:

Top-down transducer only accepting valid input trees:

\[ q_0(f(x_1, x_2)) \rightarrow f(q_a(x_2), q_b(x_1)) \]

Swap and then copy subtrees, verifying the shape of the input.

Without swapping: Transducer cannot read the right subtree of the input.
Nondeterministic Specifications

Synchronous top-down transducer: only relabels the nodes

Theorem (L./Winter). It is decidable, given an automatic tree relation, whether it can be implemented by a synchronous top-down tree transducer. (Also works for infinite trees.)
Nondeterministic Specifications

Synchronous top-down transducer: only relabels the nodes

\[
\begin{array}{c}
q \\
| \\
\downarrow f \\
\_ \\
\{} x_1 \_ x_2 \_ x_3 \\
\end{array} \rightarrow \begin{array}{c}
\_/ \\
\_ \\
\{} q_1 \_ q_2 \_ q_3 \\
\} x_1 \_ x_2 \_ x_3
\end{array}
\]

Theorem (L./Winter). It is decidable, given an automatic tree relation, whether it can be implemented by a synchronous top-down tree transducer. (Also works for infinite trees.)

Idea: Replace deterministic specification automaton in the game by a guidable automaton.

Guidable automata:

- Can “simulate” all tree automata for the same language.
- For every tree automaton there is an equivalent guidable tree automaton.
Outline

1. Classical Setting – Büchi-Landweber Theorem
2. Transducer Synthesis for Finite Words
3. Trees
4. Beyond Finite Automata
Beyond Finite Automata

Use pushdown automata on words (finite automata + stack) instead of finite automata (models finite state programs with recursion).

Synchronous case:

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 transducer.
Beyond Finite Automata

Synchronous specifications, asynchronous realizations:

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

$$
\begin{pmatrix}
0 \\
0
\end{pmatrix}^\omega \quad \text{or}
$$
Beyond Finite Automata

Synchronous specifications, asynchronous realizations:

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

$\left(0\right)^\omega$ or $\left(0\right)^n \left(0\right)^n \left(1\right) \left(I\right)^\omega$ or

$\left(1\right) \left(I\right)^\omega$.
Beyond Finite Automata

Synchronous specifications, asynchronous realizations:

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

\[
\begin{align*}
(0)^\omega & \quad \text{or} \quad (0)^n (0)^n (1) (I)^\omega \\
(0)^n (1) (I) & \quad \text{or} \quad (0)^n (0)^{n+1} (1) (I) (J)^\omega
\end{align*}
\]
Synchronous specifications, asynchronous realizations:

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

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

There is a strategy with linear delay:

$I$

$J$
Beyond Finite Automata

Synchronous specifications, asynchronous realizations:

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

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

There is a strategy with linear delay:

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

Synchronous specifications, asynchronous realizations:

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

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

There is a strategy with linear delay:

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

Synchronous specifications, asynchronous realizations:

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

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

There is a strategy with linear delay:

$I \quad 0 \quad 0$

$J$
Beyond Finite Automata

Synchronous specifications, asynchronous realizations:

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

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

There is a strategy with linear delay:

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

Synchronous specifications, asynchronous realizations:

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

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

There is a strategy with linear delay:

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

Synchronous specifications, asynchronous realizations:

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:

$I \quad 0 \quad 0 \quad 0 \\
J \quad 0 \quad \text{skip}$
Beyond Finite Automata

Synchronous specifications, asynchronous realizations:

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:

\[
\begin{align*}
I & \quad 0 \quad 0 \quad 0 \quad 0 \\
J & \quad 0 \\
\end{align*}
\]
Synchronous specifications, asynchronous realizations:

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

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

There is a strategy with linear delay:

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

Synchronous specifications, asynchronous realizations:

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

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

There is a strategy with linear delay:

\[
I \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \\
J \quad 0 \quad 0
\]
Beyond Finite Automata

Synchronous specifications, asynchronous realizations:

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

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

There is a strategy with linear delay:

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

Synchronous specifications, asynchronous realizations:

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

1. $(0^\omega)$ or $(0^n 1^n (IJ) (IJ)^\omega)$
2. $(0^n 1^{n+1} (IJ) (IJ)^\omega)$

There is a strategy with linear delay:

$I$ 0 0 0 0 0 0 1

$J$ 0 0
Beyond Finite Automata

Synchronous specifications, asynchronous realizations:

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

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

There is a strategy with linear delay:

\[
\begin{align*}
I & 0 0 0 0 0 0 1 \cdots \\
J & 0 0 1 \cdots
\end{align*}
\]
Theorem (Fridman/L./Zimmerman’11).

- There are deterministic pushdown specifications for which there is a winning strategy for the output player in the asynchronous game, but each such strategy needs non-elementary delay.
- For deterministic pushdown specifications it is undecidable if there is a winning strategy for player output in the asynchronous game.
Conclusion

Automatic synthesis of transducers from specifications

- decidability results for synchronous specifications (words and trees)
- undecidability for asynchronous specifications

Perspectives:

- Identify decidable subclasses of asynchronous specifications
- Trees: nondeterministic specifications, other transducer models