Transducer Theory and Streaming Transformations

Emmanuel Filiot

Université Libre de Bruxelles
**Finite State Automata**

- finite string acceptors over a finite alphabet $\Sigma$
- read-only input tape, left-to-right
- finite set of states

**Definition (Finite State Automaton)**

A finite state automaton (FA) on $\Sigma$ is a tuple $A = (Q, I, F, \delta)$ where

- $Q$ is the set of states,
- $I \subseteq Q$, reps. $F \subseteq Q$ is the set of initial, resp. final, states,
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition relation.

$L(A) = \{ w \in \Sigma^* \mid \text{there exists an accepting run on } w \}$
Finite State Automata – Example

![Finite State Automata Diagram](image)

**Finite State Automata**

Example:

- States: `q0` and `q1`
- Transitions:
  - `a` from `q0` to `q1`
  - `b` from `q1` to `q0`
  - `a` from `q0` to `q0`
  - `b` from `q1` to `q1`

**Language**:

- `L(A) = \{ w \in \Sigma^* | w` contains an even number of `a` \}`

Run on `aabaa`:

- Start at `q0`.
- Read `a` to `q0`.
- Read `b` to `q1`.
- Read `a` to `q0`.
- Read `b` to `q1`.
- Read `a` to `q0`.
- Read `a` to `q0`.

Accepting state: `q0`.
Finite State Automata – Example

Run on $aabaa$:
Finite State Automata – Example

Run on \textit{aabaa}:

$L(A) = \{ w \in \Sigma^* \mid w \text{ contains an even number of } a \}$
Properties of FA

Expressiveness

\[ \text{FA} = \text{regular languages} = \text{MSO}[+1] = \text{regular expressions} = \ldots \]
Properties of FA

Expressiveness

FA = regular languages = MSO[+1] = regular expressions = ...

Closure Properties

- closed under Boolean operations (union, intersection, complement).
- closed under various extensions:
  - non-determinism (NFA): $\delta \subseteq Q \times \Sigma \times Q$
  - two-way input head (2NFA): $\delta \subseteq Q \times \Sigma \times \{-1, 0, 1\} \times Q$
  - regular look-ahead: $\delta \subseteq Q \times \Sigma \times \text{Reg} \times Q$
  - alternation: $\delta : Q \times \Sigma \to B(Q)$ (Boolean formulas over $Q$)
Properties of FA

Expressiveness

FA = regular languages = MSO[+1] = regular expressions = ...

Closure Properties

- closed under Boolean operations (union, intersection, complement).
- closed under various extensions:
  - non-determinism (NFA): $\delta \subseteq Q \times \Sigma \times Q$
  - two-way input head (2NFA): $\delta \subseteq Q \times \Sigma \times \{-1, 0, 1\} \times Q$
  - regular look-ahead: $\delta \subseteq Q \times \Sigma \times \text{Reg} \times Q$
  - alternation: $\delta : Q \times \Sigma \rightarrow B(Q)$ (Boolean formulas over $Q$)

Decision Problems

Membership, emptiness, universality, inclusion, equivalence ... are decidable.
From Languages to Transductions

Let $\Sigma$ and $\Delta$ be two finite alphabets.

**Definition**

<table>
<thead>
<tr>
<th>Language on $\Sigma$</th>
<th>Transduction from $\Sigma$ to $\Delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>function from $\Sigma^*$ to ${0, 1}$</td>
<td>relation $R \subseteq \Sigma^* \times \Delta^*$</td>
</tr>
<tr>
<td>defined by automata</td>
<td>defined by transducers</td>
</tr>
<tr>
<td>accept strings</td>
<td>transform strings</td>
</tr>
</tbody>
</table>

transducer = [automaton] + output mechanism.
Finite State Transducers
Finite State Transducers

- read-only left-to-right input head
- write-only left-to-right output head
- finite set of states
Finite State Transducers

- read-only left-to-right input head
- write-only left-to-right output head
- finite set of states

**Definition (Finite State Transducers)**

A finite state transducer from $\Sigma$ to $\Delta$ is a pair $T = (A, O)$ where

- $A = (Q, I, F, \delta)$ is the underlying automaton
- $O$ is an output morphism from $\delta$ to $\Delta^*$.

- If $t = q \xrightarrow{a} q' \in \delta$, then $O(t)$ defines its output.
- $q \xrightarrow{a|w} q'$ denotes a transition whose output is $w \in \Delta^*$. 
Finite State Transducers

- read-only left-to-right input head
- write-only left-to-right output head
- finite set of states

**Definition (Finite State Transducers)**

A **finite state transducer** from $\Sigma$ to $\Delta$ is a pair $T = (A, O)$ where

- $A = (Q, I, F, \delta)$ is the **underlying automaton**
- $O$ is an **output** morphism from $\delta$ to $\Delta^*$.

If $t = q \xrightarrow{a} q' \in \delta$, then $O(t)$ defines its **output**.

$q \xrightarrow{a|w} q'$ denotes a transition whose output is $w \in \Delta^*$.

**Two classes of transducers:**
- **DFT** if $A$ is deterministic
- **NFT** if $A$ is non-deterministic.
Some applications

- language and speech processing (e.g. see work by Mehryar Mohri)
- model-checking infinite state-space systems\(^1\)
- verification of web sanitizers\(^2\)
- string pattern matching

\(^2\) see BEK, developed at Microsoft Research
Finite State Transducers – Example 1

Finite State Transducers (FSTs) are a type of automaton that can output as well as accept inputs. They are used in various applications such as natural language processing, speech recognition, and text-to-speech conversion.

Consider the following FST example:

- **States**: $q_0$ (initial state), $q_1$
- **Transitions**:
  - $q_0 \xrightarrow{a} q_0$
  - $q_0 \xrightarrow{b} q_1$
  - $q_1 \xrightarrow{b} q_1$
  - $q_1 \xrightarrow{a} q_0$

Let's run an input of $aabaa$ through this FST:

1. Start at $q_0$.
2. Read $a$: $q_0 \rightarrow q_0$.
3. Read $a$: $q_0 \rightarrow q_0$.
4. Read $b$: $q_0 \rightarrow q_1$.
5. Read $a$: $q_1 \rightarrow q_0$.
6. Read $a$: $q_1 \rightarrow q_0$.
7. Read $a$: $q_1 \rightarrow q_1$.
8. Read $a$: $q_1 \rightarrow q_1$.

The output is $aaaab$. 

This example demonstrates how an FST can transform input strings into output strings.
Finite State Transducers – Example 1

Run on $aabaa$:

$$T(aabaa) = a.a.\epsilon.a.a = aaaa.$$
Finite State Transducers – Example 1

Finite State Transducers

Example 1

\[ \begin{array}{c}
q_0 \quad b | \epsilon \\
\text{start} \quad a | a \\
q_1 \quad b | \epsilon \\
\end{array} \]

Run on \( aaba \):

\( T(aaba) = \text{undefined} \)
Finite State Transducers – Example 1

Run on $aaba$:

$$T(aaba) = \text{undefined}$$
Finite State Transducers – Example 1

Semantics

\[ \text{dom}(T) = \{ w \in \Sigma^* \mid \#_a w \text{ is even} \} \]

\[ R(T) = \{(w, a\#_a w) \mid w \in \text{dom}(T)\} \]
Finite State Transducers – Example 2

\[
\text{\(\square\) = white space}
\]

\[
\begin{array}{c}
q_0 \\
\text{start}
\end{array} \quad \xrightarrow{a|a} \quad \begin{array}{c}
q_1 \\
\text{\(\square\)|\(\epsilon\)}
\end{array}
\]

Semantics
Replace blocks of consecutive white spaces by a single white space.

\[T(aa) = aa\]
Finite State Transducers – Example 2

\[ \_ = \text{white space} \]

![Finite State Transducer Diagram]

**Semantics**

Replace blocks of consecutive white spaces by a single white space.

\[ T(\_a\_a\_a\_a\_a\_a) = \_a\_a\_a \]
Finite State Transducers – Example 3

ṣ = white space

[Diagram of a finite state transducer with states q0, q1, and q2, transitions labeled with symbols such as ε and a|a, and transitions between states.]
Finite State Transducers – Example 3

\( = \) white space

Semantics
Replace blocks of consecutive white spaces by a single white space and remove the last white spaces (if any).

\[ T(\_\_a\_a\_a\_a\_a) = \_a\_a\_a \]
Semantics

Replace blocks of consecutive white spaces by a single white space and remove the last white spaces (if any).

$$T(\quad aa\quad aa\quad) = \quad aa\quad a$$

Non-deterministic but still defines a function: functional NFT
Is non-determinism needed?
Is non-determinism needed?
How to get a deterministic FT?

- extend automata subset construction with outputs
- output the longest common prefix
How to get a deterministic FT?

- extend automata subset construction with outputs
- output the longest common prefix

\[
\begin{array}{c}
q_2 \xrightarrow{\epsilon} q_0 \\
q_0 \xrightarrow{\epsilon} q_1
\end{array}
\]
How to get a deterministic FT?

- extend automata subset construction with outputs
- output the longest common prefix
How to get a deterministic FT?

- extend automata subset construction with outputs
- output the longest common prefix
How to get a deterministic FT?

- extend automata subset construction with outputs
- output the longest common prefix
How to get a deterministic FT?

- extend automata subset construction with outputs
- output the longest common prefix
How to get a deterministic FT?

1. Extend automata subset construction with outputs.
2. Output the longest common prefix.

Diagram:

- State diagram showing transitions and outputs.
- States labeled with outputs: $q_0(a)$, $q_1(\varepsilon)$, $q_2(\varepsilon)$.
How to get a deterministic FT?

- extend automata subset construction with outputs
- output the longest common prefix
Can we always get an equivalent deterministic FT?
Can we always get an equivalent deterministic FT?

- not in general: DFT define functions, NFT define relations
- what about functional NFT?
Can we always get an equivalent deterministic FT?

- not in general: DFT define functions, NFT define relations
- what about functional NFT?

![Diagram of a finite state transducer](image)

Semantics:

\[ R(T) : \begin{cases} 
  a^n b \rightarrow b^{n+1} \\
  a^n c \rightarrow c^{n+1} 
\end{cases} \]

Functional but not determinizable
Subset construction fails ...

Subset construction:

$q_0$
Subset construction fails ...

Subset construction:
Subset construction fails ...

Subset construction:
Subset construction fails ...

**Subset construction:**
Subset construction fails ...

Subset construction:
Subset construction fails ...

\[ q_0 \xrightarrow{a|\epsilon} q_1(b) \xrightarrow{a|\epsilon} q_1(bb) \xrightarrow{a|\epsilon} q_1(bbb) \xrightarrow{a|\epsilon} \ldots \]

\[ q_2(c) \xrightarrow{a|\epsilon} q_2(cc) \xrightarrow{a|\epsilon} q_2(ccc) \]
How to guarantee termination of subset construction?

**LAG**

\[ LAG(u, v) = (u', v') \] such that \( u = \ell u', \ v = \ell v' \) and \( \ell = lcp(u, v) \).

E.g. \( LAG(abbc, abc) = (bc, c) \).
How to guarantee termination of subset construction?

**LAG**

\[ \text{LAG}(u, v) = (u', v') \text{ such that } u = \ell u', \ v = \ell v' \text{ and } \ell = \text{lcp}(u, v). \]

E.g. \( \text{LAG}(abbc, abc) = (bc, c). \)

**Lemma (Twinning Property)**

*Subset construction terminates iff for all such situations*

\[ u_1|v_1 \quad \rightarrow \quad \eta_0 \quad \rightarrow \quad q \quad \rightarrow \quad q_2 \quad \rightarrow \quad q \]

\[ u_2|v_2 \quad \rightarrow \quad q \quad \rightarrow \quad q_2 \quad \rightarrow \quad q \]

\[ u_1|w_1 \quad \rightarrow \quad \eta_0 \quad \rightarrow \quad p \quad \rightarrow \quad p_2 \quad \rightarrow \quad p \]

\[ u_2|w_2 \quad \rightarrow \quad p \quad \rightarrow \quad p_2 \quad \rightarrow \quad p \]

*it is the case that \( \text{LAG}(v_1, w_1) = \text{LAG}(v_1 v_2, w_1 w_2). \)*
Theorem (Choffrut 77, Beal Carton Prieur Sakarovitch 03)

Given a functional NFT $T$, the following are equivalent:

1. it is determinizable
2. the twinning property holds.

Moreover, the twinning property is decidable in PTime.

Proof.

Intuition

- If TP holds, then subset construction terminates and produces an equivalent DFT
- for the converse, uses the fact that TP is machine-independent: for all $T \equiv T'$, $T \models TP$ iff $T' \models TP$. 
Determinizability is decidable

**Theorem (Choffrut 77, Beal Carton Prieur Sakarovitch 03)**

*Given a functional NFT $T$, the following are equivalent:*

1. *it is determinizable*
2. *the twinning property holds.*

*Moreover, the twinning property is decidable in PTime.*

**Proof.**

**Intuition**

- If TP holds, then subset construction terminates and produces an equivalent DFT
- for the converse, uses the fact that TP is machine-independent: for all $T \equiv T'$, $T \models TP$ iff $T' \models TP$.  

Almost true ...
subsequential transducers are deterministic but can output a string in each accepting states

in the previous theorem: “determinizable” $\Leftrightarrow$ “there exists an equivalent subsequential transducer”

subsequential transducers $\equiv$ DFT if last string symbol is unique
Application: analysis of streaming transformations

**Bounded Memory Problem**

**Hypothesis:**
- input string is received as a (very long) stream
- output string is produced as a stream

**Input:** a transformation defined by some functional NFT

**Output:** can I realize this transformation with bounded memory?

\[ \exists B \in \mathbb{N} \cdot \forall u \in \text{dom}(T) \]

\[ T(u) \text{ can be computed with } B\text{-bounded memory?} \]
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)

1 0 0 1 1 1 #
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)

Working Tape (read/write)
Streaming Model

Deterministic Turing Transducer

Input Tape (read only) 1 0 0 1 1 1 1 #

Working Tape (read/write) 1 1 0 0 1 1 # #
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)  

Working Tape (read/write)
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)

Working Tape (read/write)
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)

[Diagram showing a tape with symbols: 1 0 0 1 1 1 #]

Working Tape (read/write)

[Diagram showing a tape with symbols: 1 0 0 0 0 1 # #]

---

Finite State Transducers

Extensions of NFT

VPTs

Church Problem

Conclusion

22 / 63
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)

1 0 0 1 1 1 1 #

Working Tape (read/write)

1 0 0 0 0 1 # #
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)

```
1 0 0 1 1 1 #
```

Working Tape (read/write)

```
1 0 0 0 1 1 # #
```
Streaming Model

Deterministic Turing Transducer

- **Input Tape (read only)**: 
  - `1 0 0 1 1 1 #`

- **Working Tape (read/write)**: 
  - `1 0 0 0 0 1 0 #`
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)

1 0 0 1 1 1 #

Working Tape (read/write)

1 0 0 0 0 1 0 #

Output Tape (write only)

0 # # # # # # #
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)

1 0 0 1 1 1 #

Working Tape (read/write)

1 0 0 0 0 1 0 #

Output Tape (write only)

0 1 # # # # #
**Streaming Model**

**Deterministic Turing Transducer**

- **Input Tape (read only)**
  
- **Working Tape (read/write)**
  
- **Output Tape (write only)**
Streaming Model

Deterministic Turing Transducer

- **Input Tape (read only)**: 1 0 0 1 1 1 #
- **Working Tape (read/write)**: 1 0 0 0 0 1 0 #
- **Output Tape (write only)**: 0 1 1 0 # # #
Streaming Model

Deterministic Turing Transducer

---

Input Tape (read only):

```
1 0 0 1 1 1 #
```

---

Working Tape (read/write):

```
1 0 0 0 0 1 0 #
```

---

Output Tape (write only):

```
0 1 1 0 1 # #
```
Streaming Model

Deterministic Turing Transducer

- **Input Tape**: Read only

- **Working Tape**: Read/write

- **Output Tape**: Write only
Streaming Model

Deterministic Turing Transducer

- **Input Tape** (read only):
  - 1 0 0 1 1 1 #

- **Working Tape** (read/write):
  - 1 0 0 0 0 1 0 #
  - Memory Measured on this tape only!

- **Output Tape** (write only):
  - 0 1 1 0 1 1 #
Bounded Memory Problem – Examples

\[ T_1 : \begin{cases} 
  a^n b \mapsto b^{n+1} \\
  a^n c \mapsto c^{n+1} 
\end{cases} \]

Not bounded memory

\[ T_2 : \underbrace{a \ldots a \ldots b \ldots} \mapsto \underbrace{a \ldots a \ldots b} \]

Bounded memory
Bounded Memory Problem – Examples

\[ T_1 : \begin{align*}
& a^n b \mapsto b^{n+1} \\
& a^n c \mapsto c^{n+1}
\end{align*} \]

Not bounded memory

\[ T_2 : \underline{\_\_\_\_a\_\_\_\_b\_\_\_\_ \mapsto \_\_\_\_a\_\_\_b} \]

Bounded memory

**Theorem**

For all functional NFT \( T \), the following are equivalent:

1. \( T \) is bounded memory
2. \( T \) is determinizable
3. \( T \) satisfies the twinning property.

Proof based on the following two observations:

1. any DFT is bounded memory
2. bounded memory Turing Transducer \( \equiv \) DFT
Closure Properties of Finite State Transducers

Domain, co-domain

The domains and co-domains of NFT are regular.

<table>
<thead>
<tr>
<th></th>
<th>$T^{-1}$</th>
<th>$\overline{T}$</th>
<th>$T_1 \cup T_2$</th>
<th>$T_1 \cap T_2$</th>
<th>$T_1 \circ T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFT</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>DFT</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Table: Closure Properties for NFT and DFT.
Closure Properties of Finite State Transducers

Domain, co-domain

The domains and co-domains of NFT are regular.

<table>
<thead>
<tr>
<th></th>
<th>$T^{-1}$</th>
<th>$\overline{T}$</th>
<th>$T_1 \cup T_2$</th>
<th>$T_1 \cap T_2$</th>
<th>$T_1 \circ T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFT</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>DFT</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table: Closure Properties for NFT and DFT.

Non-closure by intersection

1. $R(T_1) = \{(a^m b^n, c^m) | m, n \geq 0\}$
2. $R(T_2) = \{(a^m b^n, c^n) | m, n \geq 0\}$
3. $R(T_1) \cap R(T_2) = \{(a^n b^n, c^n) | n \geq 0\}$
Decision problems

Membership \((u, v) \in R(T)\)?

Emptiness \(R(T) = \emptyset\)?

Type checking \(T(L_{in}) \subseteq L_{out}\)?

Equivalence \(R(T_1) = R(T_2)\)?

Inclusion \(R(T_1) \subseteq R(T_2)\)?

<table>
<thead>
<tr>
<th></th>
<th>emptiness / emptiness</th>
<th>type checking / type checking</th>
<th>equiv / membership (vs NFA)</th>
<th>inclusion / inclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFT</td>
<td>PTIME</td>
<td>PSPACE-c</td>
<td>undec</td>
<td></td>
</tr>
<tr>
<td>DFT</td>
<td>PTIME</td>
<td>PSPACE-c</td>
<td>PTIME</td>
<td></td>
</tr>
</tbody>
</table>

Table: Decision problems for NFT and DFT.

Undecidability of equivalence and inclusion proved in [Griffiths68].
A transduction (transducer) is **functional** if each word has at most 1 image.

**Theorem (Gurari and Ibarra 83)**

*Functionality is decidable in PTIME for NFT.*

**Theorem**

The equivalence and inclusion of **functional** NFT is PSPACE-c.

**Proof.**

$T_1$ is included in $T_2$ if and only if

- $\text{dom}(T_1) \subseteq \text{dom}(T_2)$, and
- $T_1 \cup T_2$ is functional.
A transduction (transducer) is $k$-valued if each word has at most $k$ images.

**Theorem (GI83, Web89, SdS08)**

Let $k \in \mathbb{N}$ be fixed.

$k$-valuedness is decidable in $\text{PTime}$ for NFT.

**Theorem (IK86, Web88)**

The equivalence and inclusion of $k$-valued NFT are $\text{PSpace}$-c.
Extensions of NFT
Extensions of NFT

Various more expressive extensions have been considered:

1. two-way input tape
2. string variables (Alur Cerny 2010)
3. pushdown stack
Two-way finite state transducers (2NFT)

Input Tape

\[ \alpha|\epsilon, +1 \]

\[ \alpha|\alpha, -1 \]

\[ \epsilon|\epsilon, -1 \]

\[ \epsilon|\epsilon \]

Output Tape

head
Two-way finite state transducers (2NFT)

Input Tape

\[ \alpha | \epsilon, +1 \quad \alpha | \alpha, -1 \]

Output Tape

Head

\[ \Downarrow s \quad t \quad r \quad e \quad s \quad s \quad e \quad d \quad \Uparrow \]

\[ \Downarrow | \epsilon, -1 \quad \Downarrow | \epsilon, 1 \quad \Downarrow | \epsilon \]

Head
Two-way finite state transducers (2NFT)

Input Tape

```
|<s|t|r|e|s|s|e|d|>
```

Output Tape

```

```

α|ε, +1

1

→

-|ε, -1

2

→

|ε

3

α|ε, +1

α|α, -1

head

head
Two-way finite state transducers (2NFT)

Input Tape

\[ \alpha | \epsilon, +1 \quad \alpha | \alpha, -1 \]

Output Tape

\[ \text{head} \]
Two-way finite state transducers (2NFT)

Input Tape

Output Tape

\[\alpha | \epsilon, +1\]

\[\alpha | \alpha, -1\]

\[\alpha | \epsilon, -1\]

\[\alpha | \epsilon\]
Two-way finite state transducers (2NFT)

Input Tape

\[ \alpha | \epsilon, +1 \]

\[ 1 \]

\[ -| \epsilon, -1 \]

\[ 2 \]

\[ \alpha | \alpha, -1 \]

\[ -| \epsilon, -1 \]

\[ 3 \]

\[ \alpha | \epsilon, +1 \]

\[ 1 \]

\[ -| \epsilon, -1 \]

\[ 2 \]

\[ \alpha | \alpha, -1 \]

\[ -| \epsilon, -1 \]

\[ 3 \]

Output Tape
Two-way finite state transducers (2NFT)

Input Tape

\[\begin{array}{c}
\alpha \mid \epsilon, +1 \\
\downarrow \\
1 \\
\end{array} \quad \begin{array}{c}
\alpha \mid \alpha, -1 \\
\downarrow \\
2 \\
\end{array} \quad \begin{array}{c}
\perp \mid \epsilon \\
\downarrow \\
3 \\
\end{array}\]

Output Tape

\[\begin{array}{c}
\text{head}
\end{array}\]

\[\begin{array}{c}
\perp \\
\downarrow \\
\text{head}
\end{array}\]
Two-way finite state transducers (2NFT)

Input Tape

\[ \alpha|\epsilon, +1 \]
\[ \alpha|\alpha, -1 \]

Output Tape
Two-way finite state transducers (2NFT)

Input Tape

\[ \alpha | \epsilon, +1 \]

\[ 1 \rightarrow \neg | \epsilon, -1 \]

\[ \alpha | \alpha, -1 \]

\[ 2 \rightarrow \neg | \epsilon, -1 \]

\[ \neg | \epsilon, -1 \]

\[ 3 \]

Output Tape

head

head
Two-way finite state transducers (2NFT)

Input Tape

\[
\begin{array}{cccccccc}
| & s & t & r & e & s & s & e & d & | \\
\end{array}
\]

\[\alpha|\epsilon, +1\]
\[\neg|\epsilon, -1\]

\[\alpha|\alpha, -1\]

Output Tape

\[\text{head}\]

\[\text{head}\]
Two-way finite state transducers (2NFT)

Input Tape

1
\[ \alpha | \epsilon, +1 \]
\[ \vdash | \epsilon, -1 \]
\[ \vdash | \epsilon \]

2
\[ \alpha | \alpha, -1 \]

3

Output Tape

\[ d \]

head
Two-way finite state transducers (2NFT)

Input Tape

\[
\begin{array}{ccccccccc}
\vdash & s & t & r & e & s & s & e & d & \vdash \\
\end{array}
\]

head

\[
\begin{array}{c}
\alpha | \epsilon, +1 \\
1
\end{array}
\]

\[
\begin{array}{c}
\vdash | \epsilon, -1 \\
2
\end{array}
\]

\[
\begin{array}{c}
\alpha | \alpha, -1 \\
3
\end{array}
\]

Output Tape

\[
\begin{array}{cccccccc}
d & e & \vdash & \vdash & \vdash & \vdash & \vdash & \vdash \\
\end{array}
\]

head
Two-way finite state transducers (2NFT)

Input Tape

\[ \alpha|\epsilon, +1 \]
\[ \alpha|\alpha, -1 \]
\[ -|\epsilon, -1 \]
\[ |\epsilon \]

Output Tape

\[ d \]
\[ e \]
\[ s \]

Head
Two-way finite state transducers (2NFT)

Input Tape

\[ \alpha | \epsilon, +1 \]
\[ α | α, −1 \]

Output Tape

\[ d \quad e \quad s \quad s \quad \_ \quad \_ \quad \_ \quad \_ \quad \_ \]
Two-way finite state transducers (2NFT)

Input Tape

\[ \vdash s t r e s s e d \vdash \]

 output

\( \alpha | \epsilon, +1 \) \hspace{2cm} \( \alpha | \alpha, -1 \)

Transition

\[ 1 \rightarrow 2 \rightarrow 3 \]

Output Tape

\[ d e s s e \]

\[ \vdash \]

head
Two-way finite state transducers (2NFT)

Input Tape

```
Head

α|ε, +1

1

|→ s t r e s s e d |→
```

```
Head

α|α, −1

2

|→ ε, −1

3
```

Output Tape

```
Head

d e s s e r
```

```
\[\alpha | \epsilon, +1\] \\
\[\alpha | \alpha, -1\] \\
\[| \epsilon, -1\] \\
\[| \epsilon\] 
```
Two-way finite state transducers (2NFT)

Input Tape

\[ \begin{array}{c}
\ddownarrow & s & t & r & e & s & s & e & d & \uparrow \\
\alpha | \epsilon, +1 & \alpha | \alpha, -1 \\
\end{array} \]

Output Tape

\[ \begin{array}{c}
d & e & s & s & e & r & t \\
\downarrow & \text{head} \\
\end{array} \]
Two-way finite state transducers (2NFT)

Input Tape

\[
\begin{array}{cccccccc}
\text{s} & \text{t} & \text{r} & \text{e} & \text{s} & \text{s} & \text{e} & \text{d} & \text{1} \\
\end{array}
\]

\(\alpha|\epsilon, +1\)

\(\alpha|\alpha, -1\)

\(\text{head}\)

\[\begin{array}{c}
1 \\
\end{array}\]

Output Tape

\[\begin{array}{ccccccccc}
\text{d} & \text{e} & \text{s} & \text{s} & \text{e} & \text{r} & \text{t} & \text{s} & \text{2} \\
\end{array}\]

\(\text{head}\)
Two-way finite state transducers (2NFT)

Input Tape:

\[ \alpha | \epsilon, +1 \]

Output Tape:

\[ d | e | s | s | e | r | t | s \]
Two-way finite state transducers – Properties

Main Properties of 2NFT

1. still closed under composition (Chytil Jakl 77)
2. equivalence of functional 2NFT is decidable (Culik, Karhumaki, 87)
3. functional 2NFT $\equiv$ 2DFT (Hoogeboom Engelfriet 01, De Souza 13)

Logical Characterization (Hoogeboom Engelfriet 01)

2DFT $\equiv$ MSO transductions

2DFT define regular functions.
MSO Transductions (Courcelle)

- input string seen as the logical structure over \( \{\text{succ}, (\text{lab}_a)_{a \in \Sigma}\} \)
- output predicates defined with MSO formulas interpreted over the input structure
input string seen as the logical structure over \( \{ \text{succ}, (\text{lab}_a)_{a \in \Sigma} \} \)
output predicates defined with MSO formulas interpreted over the input structure
MSO Transductions (Courcelle)

- input string seen as the logical structure over \( \{ \text{succ}, (\text{lab}_a)_{a \in \Sigma} \} \)
- output predicates defined with MSO formulas interpreted over the input structure

\[
\phi_{\text{succ}}(x, y) \equiv \text{succ}(y, x)
\]

\[
\phi_{\text{lab}_a}(x) \equiv \text{lab}_a(x)
\]
input string seen as the logical structure over \( \{ \text{succ}, (\text{lab}_a)_{a \in \Sigma} \} \)

output predicates defined with MSO formulas interpreted over the input structure

\[
\phi_{\text{succ}}(x, y) \equiv \text{succ}(y, x)
\]

\[
\phi_{\text{lab}_a}(x) \equiv \text{lab}_a(x)
\]
MSO Transductions (Courcelle)

- input string seen as the logical structure over \( \{ succ, (lab_a)_{a \in \Sigma} \} \)
- output predicates defined with MSO formulas interpreted over the input structure

\[
\phi_{succ}(x, y) \equiv succ(y, x) \\
\phi_{lab_a}(x) \equiv lab_a(x)
\]
Streaming String Transducers (Alur, Cerny, 2010)

On every transitions, a finite set of variables can be updated by

- appending a string: \( x := x.u \)
- prepending a string: \( x := u.x \)
- concatenating two variables: \( x := yz \)
Streaming String Transducers (Alur, Cerny, 2010)

On every transitions, a finite set of variables can be updated by

- appending a string: \( x := x.u \)
- prepending a string: \( x := u.x \)
- concatenating two variables: \( x := yz \)

\[
\begin{align*}
\alpha | x & := \alpha \cdot x \\
q_0 & \quad x
\end{align*}
\]

\( R(T) = \text{mirror} \)
Streaming String Transducers (Alur, Cerny, 2010)

On every transitions, a finite set of variables can be updated by
- appending a string: \( x := x.u \)
- prepending a string: \( x := u.x \)
- concatenating two variables: \( x := yz \)

\[ R(T) = \text{mirror} \]

\[ R(T) = a^n\alpha \mapsto \alpha^{n+1} \]
Streaming String Transducers

**Theorem (Alur Cerny 2010)**

*The following models are expressively equivalent:*

1. *two-way DFT*
2. *MSO transductions*
3. *deterministic (one-way) streaming string transducers with copyless update*

Moreover, SSTs have good algorithmic properties and have been used to analyse list processing programs (Alur Cerny 2011).
Pushdown Transducers

Definition
A pushdown transducer is a pair $(A, O)$ where $A$ is a pushdown automaton and $O$ is an output morphism.

(Bad) Properties
- closure under composition is lost
- Functionality, determinizability, equivalence and inclusion of functional transducers are lost.
Finite State Transducers – Summary

\[ D = "\text{(input) deterministic}" \]
\[ f = "\text{functional}" \]

<table>
<thead>
<tr>
<th>DFTs</th>
<th>fNFTs</th>
<th>NFTs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2DFTs</td>
<td>f2NFTs</td>
<td>2NFTs</td>
</tr>
</tbody>
</table>
Finite State Transducers – Summary

D = "(input) deterministic"

f = "functional"

\[ \vdash u \iff \text{mirror}(u) \]

\[ \subset \]

\[ \subset \]

\[ \subset \]

\[ \vdash \]

\[ \dashv \mapsto \rightarrow \]

\[ \equiv \]

\[ \equiv \]

\[ \equiv \]

\[ \equiv \]

\[ \vdash \]

\[ \vdash \]

\[ \vdash \]

\[ \vdash \]

\[ \vdash \]

\[ \vdash \]
Finite State Transducers – Summary

D = “(input) deterministic”
f = “functional”
Finite State Transducers – Summary

\[ D = "(input) deterministic" \]
\[ f = "functional" \]
Finite State Transducers – Summary

D = "(input) deterministic"

f = "functional"

\[
\begin{align*}
\text{DFTs} & \subset \text{fNFTs} & \subset \text{NFTs} \\
2\text{DFTs} & \equiv & \text{f2NFTs} & \subset & 2\text{NFTs}
\end{align*}
\]

\[
\begin{align*}
\equiv & \text{MSOT [Engelfriet, Hoogeboom (01)]} \\
\equiv & \text{Streaming String Transducers [Alur, Černý, 2010]}
\end{align*}
\]

\[\text{De Souza (13)}\]
Finite State Transducers – Summary

D=“(input) deterministic”
f=“functional”

\[\text{PTIME} \subseteq \text{DFTs} \subseteq \text{fNFTs} \subseteq \text{NFTs}\]

\[\text{2DFTs} \equiv \text{f2NFTs} \subseteq \text{2NFTs}\]

\[\equiv \text{MSOT [Engelfriet, Hoogeboom (01)]}\]

\[\equiv \text{Streaming String Transducers [Alur, Černý, 2010]}\]
Finite State Transducers – Summary

D = "(input) deterministic"
f = "functional"

PTIME

\[ \text{DFTs} \subset \text{fNFTs} \subset \text{NFTs} \]

\[ \text{2DFTs} \equiv \text{f2NFTs} \subset \text{2NFTs} \]

\[ \equiv \text{MSOT [Engelfriet, Hoogeboom (01)]} \]

\[ \equiv \text{Streaming String Transducers [Alur, Černý, 2010]} \]

\[ \equiv \text{PTIME [Choffrut (77)]} \]

\[ \equiv \text{PTIME [Schützenberger (75)]} \]

\[ \equiv \text{PTIME [Gurari, Ibarra (83)]} \]

\[ \equiv \text{PTIME [Beal, Carton, Prieur, Sakarovitch (03)]} \]

\[ \equiv \text{PTIME [De Souza (13)]} \]
Finite State Transducers – Summary

D =”(input) deterministic”

f =”functional”

PTIME

[Choffrut (77)]
[Weber, Klemm (95)]
[Beal, Carton, Prieur, Sakarovitch (03)]

DFTs ⊂ fNFTs ⊂ NFTs

PTIME

[Schützenberger (75)]
[Gurari, Ibarra (83)]
[Beal, Carton, Prieur, Sakarovitch (03)]

? ⊂ ?

2DFTs  ≡

[ De Souza (13)]

≡ MSOT [Engelfriet, Hoogeboom (01)]
≡ Streaming String Transducers [Alur, Černý, 2010]

f2NFTs ⊂ 2NFTs

decidable

[ Culik, Karhumaki (87) ]
Finite State Transducers – Summary

D = "(input) deterministic"
f = "functional"
Finite State Transducers – Summary

D = ”(input) deterministic”
f = ”functional”

PTIME
[Choffrut (77)]
[Weber, Klemm (95)]
[Beal, Carton, Prieur, Sakarovitch (03)]

DFTs ⊂ fNFTs

PTIME
[Schützenberger (75)]
[Gurari, Ibarra (83)]
[Beal, Carton, Prieur, Sakarovitch (03)]

2DFTs ⊂ f2NFTs ⊂ 2NFTs

≡
[De Souza (13)]
≡ MSOT [Engelfriet, Hoogeboom (01)]
≡ Streaming String Transducers [Alur, Černý, 2010]

decidable
[Čulik, Karhumaki (87)]

open
A word about infinite strings

- most transducer models can be extended to (right-) infinite strings
- Büchi / Muller accepting conditions
- most of the results seen so far still hold with some complications ...

- determinization of one-way transducers: TP is too strong

- deterministic 2way < functional 2way:

\[ T : u \mapsto \begin{cases} a^\omega \text{ if infinite number of 'a'} \\ u \text{ otherwise} \end{cases} \]

- functional 2way \equiv \text{deterministic 2way + } \omega\text{-regular look-ahead}
  \equiv \omega\text{-MSO transductions} \equiv \omega\text{-SST (Alur,Filiot,Trivedi,12)}
Transducers for Nested Words (∼ Trees)
Motivations

**Streaming XML Transformations**

- XML are words with a nesting structure
- XML documents can be (very) wide but usually not deep
- In a streaming setting, not reasonable to keep the entire document in memory
- Bounded memory streaming transformations?
Motivations

Streaming XML Transformations
- XML are words with a nesting structure
- XML documents can be (very) wide but usually not deep
- In a streaming setting, not reasonable to keep the entire document in memory
- Bounded memory streaming transformations?

Visibly Pushdown Transducers (VPTs)
- Extend Visibly Pushdown Automata (Alur Madhusudan 04)
- Well-suited for streaming nested words transformations
- Bounded memory analysis for VPT transductions.
Structured Alphabet

Definition (Structured Alphabet)

A structured alphabet, $\Sigma$, is a set $\Sigma = \Sigma_c \cup \Sigma_i \cup \Sigma_r$, where

- $\Sigma_c$ are call symbols,
- $\Sigma_i$ are internal symbols,
- $\Sigma_r$, are return symbols.

- a nested word is a word over a structured alphabet

$$c_1 \ c_2 \ a \ r_1$$

- it is well-nested if there is no pending call nor return symbols

$$c_1 \ c_2 \ a \ r_2 \ b \ r_1$$
Nested Words vs Trees

Encoding

Well-nested words \(\equiv\) linearizations of trees

- nested words are well-suited to model tree streams
Visibly Pushdown Automata (VPAs) [Alur, Madhusudan, 04]

VPAs = Pushdown Automata on structured alphabet

\[ \Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i : \]

- push **one** stack symbol on **call** symbols \( \Sigma_c \)
- pop **one** stack symbol on **return** symbols \( \Sigma_r \)
- don’t touch the stack on **internal** symbols \( \Sigma_i \)
- in this talk, accept on empty stack and final state
Visibly Pushdown Automata (VPAs) [Alur, Madhusudan, 04]

VPAs = Pushdown Automata on \textbf{structured} alphabet \( \Sigma = \Sigma_c \uplus \Sigma_r \uplus \Sigma_i \):

- push \textbf{one} stack symbol on \textbf{call} symbols \( \Sigma_c \)
- pop \textbf{one} stack symbol on \textbf{return} symbols \( \Sigma_r \)
- don’t touch the stack on \textbf{internal} symbols \( \Sigma_i \)
- in this talk, accept on empty stack and final state

\[
L(A) = \{ c^n \ i \ r^n \ a \mid n > 0 \} \cup \{ c^n \ i \ r^n \ b \mid n > 0 \}
\]
Properties of VPA

- NFA $<$ VPA $<$ PA
- close under all Boolean operations
- NFA algorithmic properties are preserved (equivalence, universality, ...)
- applications in
  - computer-aided verification
  - XML processing
- see http://www.cs.uiuc.edu/~madhu/vpa/
Visibly Pushdown Transducers (VPTs)

Definition
Pair \((A, O)\) where \(A : VPA\) and \(O\) is an output morphism.

\[
R(T) = \{(c^n i r^n a, a^{2n}) \mid n > 0\} \cup \{(c^n i r^n b, b^{2n}) \mid n > 0\}
\]
Properties of Visibly Pushdown Transducers

- $\text{NFT} < \text{VPT} < \text{PT}$
- $\text{dVPTs} < \text{(functional) VPT}$
- closed under composition if the output is well-nested
- closed under VPA-lookahead
- functionality is decidable in PTime
- $k$-valuedness is decidable
- equivalence of functional VPTs is decidable (in PTime of dVPTs)
- decidable typechecking problem (if the output is well-nested)
Properties of Visibly Pushdown Transducers

- NFT $<$ VPT $<$ PT
- $d$VPTs $<$ (functional) VPT
- closed under composition if the output is well-nested
- closed under VPA-lookahead
- functionality is decidable in PTime
- $k$-valuedness is decidable
- equivalence of functional VPTs is decidable (in PTime of $d$VPTs)
- decidable typechecking problem (if the output is well-nested)

**Open Problems**: equivalence of $k$-valued VPTs, determinizability

- more details in F. Servais’s Phd thesis
Why is determinizability more difficult?

It is determinizable by:

\[ q_0^{\text{initial}} \xrightarrow{c|\epsilon, +\gamma} q_1 \xrightarrow{c|\epsilon, +\gamma} \text{initial} \xrightarrow{c|\epsilon, +\gamma} p_1 \xrightarrow{i|\epsilon} p_2 \]

but lag increase arbitrarily in \((p_1, q_1)\).
Why is determinizability more difficult?

It is determinizable by:

but lag increase arbitrarily in \((p_1, q_1)\).
**Streamability Problem** [F, Gauwin, Reynier, Servais, 11]

**Streaming evaluation:** avoid the storage of the whole input

Fix a functional (non-deterministic) VPT $T$.

How much memory is needed to compute $T(u)$ from an input stream $u$?
Streamability Problem [F, Gauwin, Reynier, Servais, 11]

**Streaming evaluation:** avoid the storage of the whole input

Fix a functional (non-deterministic) VPT T.

How much memory is needed to compute T(u) from an input stream u?

![Diagram showing memory usage and well-nestedness](image)
**Streamability Problem** [F, Gauwin, Reynier, Servais, 11]

**Streaming evaluation:** avoid the storage of the whole input

Fix a functional (non-deterministic) VPT $T$.

How much memory is needed to compute $T(u)$ from an input stream $u$?

---

A diagram illustrating the concept of memory usage:

- **constant memory**
- **$\text{height}(u)$**
- **dependent in $\text{length}(u)$**

**cannot check well-nestedness!**

**not streamable!**

---

**Streamability Problem**

Given a VPT $T$, decide if $T$ defines a transformation that can be evaluated with memory $O(f(\text{height}(u)))$?
Streamability Problem [F, Gauwin, Reynier, Servais, 11]

**Streaming evaluation**: avoid the storage of the whole input

Fix a functional (non-deterministic) VPT $T$. How much memory is needed to compute $T(u)$ from an input stream $u$?

Streamability Problem

Given a VPT $T$, decide if $T$ defines a transformation that can be evaluated with memory $O(f(\text{height}(u)))$?

Decidable in NP for VPTs
Determinizability is too strong

**Obs:** Deterministic VPTs are always streamable (no output lag)
Determinizability is too strong

**Obs:** Deterministic VPTs are always streamable (no output lag)

**However:** determinizable VPTs $<$ streamable VPTs:

$$R(T) : c^n i r^n \alpha \mapsto \alpha^{2^n} \quad n > 0$$

Streamable but not determinizable!
Definition

For all such situations

\[ u_1 | v_1 \quad \text{and} \quad u_2 | v_2 \]

\[ u_1 | w_1 \quad \text{and} \quad u_2 | w_2 \]

it is the case that \( LAG(v_1, w_1) = LAG(v_1 v_2, w_1 w_2) \).
Twinning Property for VPTs

**Definition**

For all such situations

$$\begin{align*}
(q_0, \bot) & \quad (q, \sigma) \quad (q, \sigma) \\
\quad u_1 \vert v_1 & \quad u_2 \vert v_2 \\
(q, \sigma) & \quad (p, \sigma') \quad (p, \sigma') \\
\quad u_1 \vert w_1 & \quad u_2 \vert w_2
\end{align*}$$

it is the case that $\text{LAG}(v_1, w_1) = \text{LAG}(v_1v_2, w_1w_2)$. 

\[49/63\]
Twinning Property for VPTs

stack height

input word

same lag

$U_1$ $U_2$ $q,\sigma$ $p,\sigma'$ $q,\sigma'$
Twinning Property for VPTs

- Stack height
- Input word
- Lag 1 < Lag 2
Twinning Property for VPTs

\[ U_1 \quad q, \sigma \quad p, \sigma' \quad U_2 \quad q, \sigma \quad p, \sigma' \quad U_2 \]

stack height

input word

lag1 < lag2 < lag3
Twinning Property for VPTs
Twinning Property for VPTs

**Theorem**

Given a functional VPT $T$, $T$ is streamable iff the twinning property holds. 

*It can be decided in NPtime.*
Theorem

Given a functional VPT $T$, $T$ is streamable iff the twinning property holds.

It can be decided in NPtime.

- TP is machine-independent: streamable VPTs is class of transductions.
- decidability based on reversal-bounded pushdown counter machines
- same result extend to strongly streamable (memory depends only on current height)
Other tree transducer models

- top-down tree transducers

\[ q(f(x_1, \ldots, x_n)) \rightarrow C[q_1(x_{i_1}), \ldots, q_p(x_{i_p})] \]

(see TATA\(^3\) book

- macro tree transducers

```plaintext
fun q(t1 t2 t3 t4 t)=
  if t = a() then
    return F (t1,t2)
  else
    if t=g(u,v) then
      return C(q'(t1,t2,u), q''(t3,t4,v))
```

- see Joost Engelfriet and Sebastian Maneth’s work

\(^3\) *Tree Automata Techniques and Applications*, tata.gforge.inria.fr
Church Problem
Church Problem (aka Church Synthesis Problem)

**Definition (Church 57)**

- $R$ a relation, or *requirements*, from a domain $D$ to a domain $D'$
- synthesize a program $P$ such for all $X \in D$, $(X, P(X)) \in R$. 
### Church Problem (aka Church Synthesis Problem)

**Definition (Church 57)**

- $R$ a relation, or *requirements*, from a domain $D$ to a domain $D'$
- synthesize a program $P$ such for all $X \in D$, $(X, P(X)) \in R$.

### Reactive System Synthesis

Let $\Sigma_{in}$ and $\Sigma_{out}$ be to finite alphabets.

- reactive systems continuously react to stimuli produced by some *uncontrollable* environment
- $D = \Sigma_{in}^\omega$, $D' = \Sigma_{out}^\omega$
- $R$ is a synchronous relation given by a (non-deterministic) symbol-to-symbol Büchi transducer
- $P$ is a Mealy machine (deterministic symbol-to-symbol transducer)
Reactive System Synthesis: Example

- \( \Sigma_{in} = \{ \text{req, nop} \} \)
- \( \Sigma_{out} = \{ \text{grant, nop} \} \).
- Requirement \( R \): if there is a request, it must be eventually granted

![Diagram showing a finite state machine with states q0 and q2, transitions labeled with inputs and actions.]

\( q_0 \) transitions to \( q_2 \) on req, nop, and grant.
\( q_2 \) transitions back to \( q_0 \) on grant.
\( q_0 \) transitions back to itself on nop.
\( q_2 \) transitions to \( q_2 \) on nop.
Reactive System Synthesis: Example

- $\Sigma_{in} = \{\text{req, nop}\}$
- $\Sigma_{out} = \{\text{grant, nop}\}$.
- Requirement $R$: if there is a request, it must be eventually granted

Possible programs (Mealy machines) that realize $R$:
Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.
Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in \( \Sigma_{in} \)
- Player *out* chooses output symbols in \( \Sigma_{out} \)
- they play during an infinite number of rounds.

Player in \((\Sigma_{in})\) :

Player *out* \((\Sigma_{out})\) :
**Church Game**

### Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

**Def:** Player *out* wins if $(i_1, o_2, o_3, ...)$ $\in R$.

**Prop:** There exists a program that realizes the requirements $R$ iff Player *out* has a winning strategy.
Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

Player in ($\Sigma_{in}$) : $i_1$

Player *out* ($\Sigma_{out}$) : $o_1$
Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

\[
\begin{align*}
\text{Player in} (\Sigma_{in}) & : i_1 \quad i_2 \\
\text{Player out} (\Sigma_{out}) & : o_1
\end{align*}
\]
Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

Player *in* ($\Sigma_{in}$) : $i_1$, $i_2$

Player *out* ($\Sigma_{out}$) : $o_1$, $o_2$
**Church Game**

**Definition**

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

Player *in* ($\Sigma_{in}$) : $i_1$ $i_2$ $i_3$

Player *out* ($\Sigma_{out}$) : $o_1$ $o_2$
Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

Player *in* ($\Sigma_{in}$) : $i_1$ $i_2$ $i_3$

Player *out* ($\Sigma_{out}$) : $o_1$ $o_2$ $o_3$
Church Game

**Definition**

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

$$\begin{align*}
\text{Player in (} \Sigma_{in} \text{)} & : i_1 \ i_2 \ i_3 \ i_4 \\
\text{Player out (} \Sigma_{out} \text{)} & : o_1 \ o_2 \ o_3
\end{align*}$$
Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

Player *in* ($\Sigma_{in}$) : $i_1$ $i_2$ $i_3$ $i_4$

Player *out* ($\Sigma_{out}$) : $o_1$ $o_2$ $o_3$ $o_4$
Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

Player *in* $(\Sigma_{in}) : i_1 \ i_2 \ i_3 \ i_4 \ i_5$

Player *out* $(\Sigma_{out}) : o_1 \ o_2 \ o_3 \ o_4$
Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

Player in ($\Sigma_{in}$) : $i_1 \quad i_2 \quad i_3 \quad i_4 \quad i_5$

Player out ($\Sigma_{out}$) : $o_1 \quad o_2 \quad o_3 \quad o_4 \quad o_5$
### Church Game

#### Definition
- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

\[
\begin{align*}
\text{Player in (} \Sigma_{in} \text{)} : & \quad i_1 \quad i_2 \quad i_3 \quad i_4 \quad i_5 \quad \ldots \\
\text{Player out (} \Sigma_{out} \text{)} : & \quad o_1 \quad o_2 \quad o_3 \quad o_4 \quad o_5 \quad \ldots
\end{align*}
\]
Church Game

Definition

- **turn-based** game between two players
- Player **in** chooses input symbols in $\Sigma_{in}$
- Player **out** chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

Player in $(\Sigma_{in})$ : $i_1 \ i_2 \ i_3 \ i_4 \ i_5 \ \ldots$

Player out $(\Sigma_{out})$ : $o_1 \ o_2 \ o_3 \ o_4 \ o_5 \ \ldots$

- **Def:** Player **out** wins if $(i_1i_2i_3\ldots, o_1o_2o_3\ldots) \in R$. 
**Church Game**

### Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in $\Sigma_{in}$
- Player *out* chooses output symbols in $\Sigma_{out}$
- they play during an infinite number of rounds.

**Player in** ($\Sigma_{in}$) : $i_1$ $i_2$ $i_3$ $i_4$ $i_5$ ... 

**Player out** ($\Sigma_{out}$) : $o_1$ $o_2$ $o_3$ $o_4$ $o_5$ ... 

- **Def:** Player *out* wins if $(i_1 i_2 i_3 \ldots, o_1 o_2 o_3 \ldots) \in R$.
- **Prop:** There exists a program that realizes the requirements $R$ iff Player *out* has a winning strategy.
State of the Art

- reactive system synthesis from $\omega$-regular specifications is decidable (Büchi Landweber 69)
- reactive system synthesis from LTL specifications is 2-ExpTime-c (Pnueli Rosner 89)
- several tools for LTL synthesis:
  - Lily (Jobstmann Bloem 06)
  - Acacia (Filiot Jin Raskin 09)
  - Unbeast (Ehlers 10)
- very active community in game theory for synthesis
  - quantitative games
  - multi-player games
  - stochastic games
  - ...
GenBuf spec from IBM

Scalable example

From 1-page long to 4-page long specifications

http://lit2.ulb.ac.be/acaciaplus
How is it related to transducer theory?

- reactive systems are streaming machines
- from a relation $R$, extract a function $f$ such that:
  1. $\text{dom}(R) \subseteq \text{dom}(f)$
  2. for all $u \in \text{dom}(R)$, $f(u) \in R(u)$.
  3. $f$ is a deterministic symbol-to-symbol transducer
- this problem is known as the uniformization problem in transducer theory
- equivalently, is there a bounded memory (symbol-to-symbol) function $f$ such that $f \subseteq R$ and $\text{dom}(R) \subseteq \text{dom}(f)$?
Conclusion
Contributions

- finite transducers have good closure and algorithmic properties
- nicely extend to visibly pushdown transducers
- streamability problem $\equiv$ synthesis problem
Open Problems and Future Work

Open problems

- equivalence of $k$-valued VPTs
- determinizability of VPTs
- extension of streaming results to more expressive transducers, e.g. macro tree transducers
- shift from reactive systems to list processing program synthesis
Publications