From Two-Way to One-Way Finite State Transducers

Emmanuel Filiot\textsuperscript{1}, Olivier Gauwin\textsuperscript{3}, P.-A. Reynier\textsuperscript{2}, Frédéric Servais\textsuperscript{1}

\textsuperscript{1}Université Libre de Bruxelles \quad \textsuperscript{2}Marseille University \quad \textsuperscript{3}Bordeaux University
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.

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

\[ L(A) = \{ w \in \Sigma^* | w \text{ contains an even number of } a \} \]
Finite State Automata – Example

Run on \textit{aabaa}:

\[ L(A) = \{ w \in \Sigma^* | \text{w contains an even number of } a \} \]
Run on \textit{aabaa}: 

\( L(A) = \{ w \in \Sigma^* \mid w \text{ contains an even number of } a \} \)
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.
One-Way 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\(^1\)
- string pattern matching
- verification of web sanitizers\(^2\)

---

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

Run on \textit{aabaa}:

\[ T(aabaa) = \text{aaaa}. \]
Finite State Transducers – Example 1

Run on $aabaa$:

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

Start

\( q_0 \)

\( q_1 \)

Transition:

- \( a \rightarrow a \) from \( q_0 \) to \( q_0 \)
- \( b \rightarrow b \) from \( q_0 \) to \( q_1 \)
- \( b \rightarrow b \) from \( q_1 \) to \( q_1 \)
- \( a \rightarrow a \) from \( q_1 \) to \( q_0 \)

Run on \( aaba \):

- Start at \( q_0 \)
- \( a \rightarrow a \)
- \( a \rightarrow a \)
- \( b \rightarrow b \)
- \( b \rightarrow b \)
- \( T(aaba) = \text{undefined} \)
Finite State Transducers – Example 1

Run on $aaba$:

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

**Semantics**

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

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

= white space

\[
\begin{align*}
q_0 & \xrightarrow{a} q_0 & q_1 & \xrightarrow{\epsilon} q_1 \\
\text{start} & \xrightarrow{a} q_0 & & \xrightarrow{a} q_1
\end{align*}
\]
Finite State Transducers – Example 2

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

\[ T(\_\text{aa}\_\_\text{a}\_\_) = \text{aa}\_\text{a}\_] \]
= white space

Finite State Transducers – Example 3

\[ T(aa) = aa \]

Non-deterministic but still defines a function: functional NFT
Finite State Transducers – Example 3

\( \_ = \text{white space} \)

\[
\begin{array}{ccc}
q_2 & \xrightarrow{\_ | \epsilon} & q_0 \\
& \xleftarrow{\_ | \epsilon} & \\
q_0 & \xrightarrow{a | a} & q_1 \\
& \xleftarrow{a | a} & \\
q_1 & \xrightarrow{\_ | \epsilon} & \\
\end{array}
\]

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

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

\[
\text{\underline{\hspace{1cm}}} = \text{white space}
\]

Semantics

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

\[
T(\text{\underline{\hspace{1cm}}}aa\text{\underline{\hspace{1cm}}}a\text{\underline{\hspace{1cm}}}) = \text{\underline{\hspace{1cm}}}aa\text{\underline{\hspace{1cm}}}a
\]

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

$q_0$
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?

- 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
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?

Semantics

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

functional but not determinizable
Subset construction fails ...

Subset construction:
Subset construction fails ...

Subset construction:
Subset construction fails ...

Subset construction:
Subset construction fails ...

Subset construction:

$q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_0$

$q_0 \rightarrow q_1(b) \rightarrow q_2(c) \rightarrow q_2(cc) \rightarrow q_1(bb) \rightarrow q_2(c) \rightarrow q_0 \rightarrow q_1(b)$
Subset construction fails ...

Subset construction:
Subset construction fails ...

Subset construction:
How to guarantee termination of subset construction?

**LAG**

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

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

\[ \text{LAG}(v_1, w_1) = \text{LAG}(v_1 v_2, w_1 w_2). \]
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.
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)

\[1\ 0\ 0\ 1\ 1\ 1\ 1\ #\]
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)

1 0 0 1 1 1 1 #
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)

| 1 | 0 | 0 | 1 | 1 | 1 | # |

Working Tape (read/write)

| 1 | 1 | 0 | 0 | 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)

0 0 1 1 1 #

Working Tape (read/write)

1 0 1 0 1 # #
Streaming Model

Deterministic Turing Transducer

Input Tape (read only)  
![Input Tape Diagram](image)

Working Tape (read/write)  
![Working Tape Diagram](image)
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 #

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 1

Working Tape (read/write): 1 0 0 0 0 1 0

Arrow indicates the direction of movement of the tape head.
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 1 0 #

Output Tape (write only)

0 1 1 0 1 1 #

Memory Measured on this tape only!
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

\[ T_2 : \ \_\_a\_\_b\_\_ \mapsto \_\_a\_b \]

Bounded memory
Bounded Memory Problem – Examples

\[ T_1 : \begin{cases} 
  a^n b \mapsto b^{n+1} \\
  a^n c \mapsto c^{n+1} 
\end{cases} \quad \text{Not bounded memory} \]

\[ T_2 : \ldots a \ldots b \ldots \mapsto \ldots a \ldots b \quad \text{Bounded memory} \]

**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
Corollary

For all transductions $R$, the following are equivalent:

1. $R$ is computable with bounded memory
2. $R$ is definable by some DFT
Two-Way Finite State Transducers

extending finite state transducers with a two-way input tape.
Two-way finite state transducers (2NFT)

Input Tape: [1NFT 2NFT 2DFA to 1NFA 2DFT to 1NFT Conclusion]

Two-way finite state transducers (2NFT)

Input Tape:

```
| ← s t r e s s e d 0 |
```

head

```
α|ε, +1
```

```
α|α, −1
```

Output Tape:

```
head
```

1 2 3
Two-way finite state transducers (2NFT)

Input Tape

Output Tape

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

Input Tape

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

head

\[ \begin{array}{cccccccc}
\alpha | \epsilon, +1 & \neg | \epsilon, -1 & \alpha | \alpha, -1 & \neg | \epsilon \\
1 & 2 & 3 & \\
\end{array} \]

Output Tape

\[ \begin{array}{cccccccc}
\leftarrow & \text{head} & \text{head} & \text{head} & \text{head} & \text{head} & \text{head} & \text{head} & \text{head} & \rightarrow \\
\end{array} \]
Two-way finite state transducers (2NFT)

Input Tape

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

head

\[
\begin{align*}
\alpha | \epsilon, +1 \\
\alpha | \alpha, -1 \\
\vdash | \epsilon, -1 \\
\vdash | \epsilon
\end{align*}
\]

Output Tape

\[
\begin{array}{cccccccccc}
\vdash & \vdash & \vdash & \vdash & \vdash & \vdash & \vdash & \vdash & \vdash & \vdash \\
\end{array}
\]

head
Two-way finite state transducers (2NFT)

Input Tape

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

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

\[ - | \epsilon, -1 \]

Output Tape

head
Two-way finite state transducers (2NFT)

Input Tape

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

Input Tape

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

\[ \text{1} \]

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

\[ \text{2} \]

\[ \text{3} \]

Output Tape

head
Two-way finite state transducers (2NFT)

Input Tape

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

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

Output Tape

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

Input Tape

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

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

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

\[ \neg|\epsilon \]

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

Input Tape: \[ s \rightarrow \alpha, +1 \]
\[ t \rightarrow \epsilon, -1 \]
\[ r \leftarrow \alpha, -1 \]
\[ e \leftarrow \epsilon \]
\[ s \leftarrow \epsilon \]
\[ s \leftarrow \epsilon \]
\[ e \leftarrow \epsilon \]
\[ d \leftarrow \epsilon \]
\[ \rightarrow \]

Output Tape: [blocks]

Head

Conclusion
Two-way finite state transducers (2NFT)

Input Tape

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

Input Tape

\[ \begin{array}{cccccc}
\alpha|\epsilon, +1 & \epsilon | -1 & \alpha | \alpha, -1 & \epsilon | 1
\end{array} \]

Output Tape

\[ \begin{array}{cccccc}
d & e & & & & & \\
\end{array} \]
Two-way finite state transducers (2NFT)

Input Tape

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

\[ 1 \]
\[ 2 \]
\[ 3 \]

Output Tape

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

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

Input Tape

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

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

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

\[ \epsilon | \epsilon \]

Output Tape

\[ d \quad e \quad s \quad s \quad \epsilon \quad \epsilon \quad \epsilon \quad \epsilon \quad \epsilon \]

head
Two-way finite state transducers (2NFT)

Input Tape

1

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

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

Output Tape

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

head
Two-way finite state transducers (2NFT)

Input Tape

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

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

Output Tape

\[ d \quad e \quad s \quad s \quad e \quad r \]

head

head
Two-way finite state transducers (2NFT)

Input Tape

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

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

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

\[ \epsilon|-1 \]

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

Input Tape

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

\[ \alpha \mid \alpha, -1 \]

\[ \epsilon \mid \epsilon, -1 \]

\[ \epsilon \mid \epsilon \]

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

\[ \alpha \mid \alpha, -1 \]

\[ -1 \mid \epsilon, -1 \]

\[ 1 \rightarrow 2 \rightarrow 3 \]

Output Tape

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

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

\[ \epsilon \mid \epsilon \]

\[ \epsilon \mid \epsilon \]

\[ \epsilon \mid \epsilon \]

\[ \epsilon \mid \epsilon \]

head

head
Two-way finite state transducers (2NFT)

Input Tape

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

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

\[|-|\varepsilon, -1\]

\[|-|\varepsilon\]

Output Tape

\[d\]

\[e\]

\[s\]

\[s\]

\[e\]

\[r\]

\[t\]

\[s\]
## Two-way finite state transducers – Properties

### Main Properties of 2NFT

1. 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)

\[ 2DFT \equiv \text{MSO transductions} \]

2DFT define regular functions.
Main Properties of 2NFT

1. 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.

Also characterized by (deterministic) streaming string transducers (Alur Cerny 10).
Motivation: bounded memory problem

Question

Is the bounded memory problem decidable for 2DFT?
Motivation: bounded memory problem

Question
Is the bounded memory problem decidable for 2DFT?

Necessary Condition
The transduction must be definable by a one-way finite state transducer.

Indeed, bounded memory computable transductions $\equiv$ 1DFT-definable transductions
Summary

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

DFTs ⊂ fNFTs ⊂ NFTs

2DFTs ⊂ f2NFTs ⊂ 2NFTs
Summary

D = "(input) deterministic"

f = "functional"

\[ \downarrow \hspace{1cm} \downarrow \hspace{1cm} \text{mirror}(u) \]

\[ \Leftrightarrow \hspace{1cm} \subseteq \hspace{1cm} \subseteq \]

\[ \left\{ (a, a), (a, b) \right\} \]

\[ \equiv \]

De Souza (13)

MSOT [Engelfriet, Hoogeboom (01)]

Streaming String Transducers [Alur,ˇCern´y, 2010]

PTIME [Choffrut (77)]

[Weber, Klemm (95)]

[Beal, Carton, Prieur, Sakarovitch (03)]

PTIME [Sch¨utzenberger (75)]

[Gurari, Ibarra (83)]

[Beal, Carton, Prieur, Sakarovitch (03)]

decidable [Culik, Karhumaki (87)]

Filiot, Gauwin, Reynier, Servais (13)

open
Summary

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

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

\[ a^n \alpha \mapsto \alpha^{n+1} \]

\[
\begin{array}{c}
\text{DFTs} \\
\subset \text{fNFTs} \\
\subset \text{NFTs} \\
\end{array}
\]

\[
\begin{array}{c}
\text{2DFTs} \\
\subset \text{f2NFTs} \\
\subset \text{2NFTs} \\
\end{array}
\]
Summary

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

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

[De Souza (13)]

\equiv MSOT [Engelfriet, Hoogeboom (01)]

\equiv Streaming String Transducers [Alur, Černý, 2010]
Summary

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

\[
\begin{align*}
P\text{TIME} & \subset \text{DFTs} \\
\text{[Choffrut (77)]} & \subset \text{fNFTs} \\
\text{[Weber, Klemm (95)]} & \subset \text{NFTs} \\
\text{[Beal, Carton, Prieur, Sakarovitch (03)]} & \subset \{ (a, a), (a, b) \} \\
\text{PTIME} & \equiv \text{f2NFTs} \\
\text{[Choffrut (77)]} & \equiv \text{MSOT [Engelfriet, Hoogeboom (01)]} \\
\text{[Beal, Carton, Prieur, Sakarovitch (03)]} & \equiv \text{Streaming String Transducers [Alur, Černý, 2010]}
\end{align*}
\]
**Summary**

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

\[ \begin{align*}
\text{PTIME} & \supseteq \text{DFTs} \\
& \supseteq \text{fNFTs} \\
& \supseteq \text{NFTs} \\
\text{2DFTs} & \equiv \text{f2NFTs} \\
\equiv & \text{MSOT [Engelfriet,Hoogeboom (01)]} \\
\equiv & \text{Streaming String Transducers [Alur, Černý, 2010]} \\
\text{PTIME} & \supseteq \text{Beal,Carton,Prieur,Sakarovitch (03)} \\
\text{PTIME} & \supseteq \text{Beal,Carton,Prieur,Sakarovitch (03)} \\
\end{align*} \]
Summary

D = "(input) deterministic"

f = "functional"

\[
\begin{array}{c}
\text{DFTs} \subset \text{fNFTs} \\
\text{NFTs}
\end{array}
\]

\[
\begin{array}{c}
\text{2DFTs} \equiv \text{f2NFTs} \\
\text{2NFTs}\equiv \text{MSOT}\equiv \text{Streaming String Transducers}
\end{array}
\]

PTIME

[Choffrut (77)]

[Weber, Klemm (95)]

[Beal, Carton, Prieur, Sakarovitch (03)]

[Schützenberger (75)]

[Gurari, Ibarra (83)]

[Beal, Carton, Prieur, Sakarovitch (03)]

PTIME

decidable

[Culik, Karhumaki (87)]
Summary

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

PTIME

DFTs ⊂ fNFTs

PTIME

NFTs

DFTs ⊂ f2NFTs ⊂ NFTs

2DFTs ≡ Streaming String Transducers [Alur, Černý, 2010]

2NFTs

f2NFTs ⊂ 2NFTs

decidable

≡ MSOT [Engelfriet, Hoogeboom (01)]

≡ Streaming String Transducers [Alur, Černý, 2010]
Summary

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

\[
\begin{align*}
\text{PTIME} & \\
\text{[Choffrut (77)]} & \\
\text{[Weber, Klemm (95)]} & \\
\text{[Beal, Carton, Prieur, Sakarovitch (03)]} & \subset \\
\text{DFTs} & \subset \\
\text{fNFTs} & \subset \\
\text{PTIME} & \\
\text{[Schützenberger (75)]} & \\
\text{[Gurari, Ibarra (83)]} & \\
\text{[Beal, Carton, Prieur, Sakarovitch (03)]} & \subset \\
\end{align*}
\]

\[
\begin{align*}
\text{2DFTs} & \equiv \\
\text{[De Souza (13)]} & \\
\text{MSOT [Engelfriet, Hoogeboom (01)]} & \\
\text{Streaming String Transducers [Alur, Černý, 2010]} & \\
\text{f2NFTs} & \subset \\
\text{decidable} & \equiv \\
\text{[Culik, Karhumaki (87)]} & \\
\end{align*}
\]
From Two-Way to One-Way Finite State Automata
Theorem

For every 2NFA there exists an equivalent 1NFA.

- The first proof was done by Rabin and Scott (1959).
- In the same journal Shepherdson (1959) also published a (simpler) proof. Also rephrased, in an even simpler way, by Ullman.
- Vardi (1981) presented a different proof.
- The R&S proof is more easily adapted to transducers.
Rabin and Scott’s proof for 2-Automata

- a run is made of many zigzags (moves of the input head)
- a run is made of many zigzags (moves of the input head)

- A \textit{z-motion} is an elementary zigzag.
Rabin and Scott’s Proof: \( z \)-motions removal

Def: A shape is \( k \)-crossing if any position is visited at most \( k \) times.

Thm: Any \( k \)-crossing shape can be reduced to a line in \( k^2 \) steps.

Prop: \( \forall w \in L(A), w \) is accepted by a \( |Q| \)-crossing run.
Def: A shape is $k$ crossing if any position is visited at most $k$ times.

Thm: Any $k$-crossing shape can be reduced to a line in $k^2$ steps.

Prop: $\forall w \in L(A)$, $w$ is accepted by a $|Q|$ crossing run.
$S = \text{squeeze}(A)$ removes some $z$-motions of $A$.

1. simulates $A$
2. non-deterministically guesses that a $z$-motion starts (e.g. from $q_1$ to $q_2$)
3. **checks** that is indeed a $z$-motion and **simulates** it one-way
4. goes back to mode 1
\[ S = \text{squeeze}(A) \] removes some \( z \)-motions of \( A \).

**Iterate squeeze(\( A \))**

- Every accepted word has a one-way run in \( \text{squeeze}^{|Q|^2}(A) \)
  \[ \implies \text{remove backward transitions to obtain a 1NFA equivalent to } A. \]
How to simulate a $z$-motion run in one-way?

Simulate the three passes in parallel! (with triple of states)
Extension to transducers

- same canvas (Rabin and Scott)
- removal of z-motion:
  - translate a z-motion transducer into a fNFT
- not always possible → decision procedure
Extension to transducers

- same canvas (Rabin and Scott)
- removal of $z$-motion:
  - translate a $z$-motion transducer into a fNFT
- not always possible $\rightarrow$ decision procedure

Remarks:
- if local $z$-motion transductions are 1-way definable, then $\text{squeeze}(T)$ can be defined
- iterate $\text{squeeze}(T) \mid Q \mid^2$ times (if possible), you get an equivalent 1-NFT
Extension to transducers

- same canvas (Rabin and Scott)
- removal of $z$-motion:
  - translate a $z$-motion transducer into a $fNFT$
- not always possible $\rightarrow$ decision procedure

Remarks:
- if local $z$-motion transductions are 1-way definable, then $\text{squeeze}(T)$ can be defined
- iterate $\text{squeeze}(T) |Q|^2$ times (if possible), you get an equivalent 1-NFT

Results:
- decision procedure to test whether a $z$-motion-transducer is 1-way definable
- the algorithm is complete
Decision procedure

Let $T$ be a f2NFT.

1. Repeat $|Q|^2$ times:
   - Are all $z$-motion transductions of $T$ NFT-definable?
     - Yes: $T \leftarrow \text{squeeze}(T)$
     - No: STOP: the initial 2NFT was not NFT-definable!

2. Remove backward transitions: you get an equivalent NFT
Towards a characterization of 1-way definable z-motion-transductions

\[
x_1 \cdot \alpha^n \cdot y_1 \cdot \beta^n \cdot x_2 \cdot \gamma^n \cdot y_2 = x_0 \cdot \delta^n \cdot y_0
\]

Lemma (Fine and Wilf (56))

Let \( u, v \in \Sigma^* \). If \( u^\omega \) and \( v^\omega \) have a sufficiently large common factor, then \( u \in (w_1 w_2)^* \) and \( v \in (w_2 w_1)^* \) for some \( w_1, w_2 \in \Sigma^* \).

\[\Rightarrow\] \( \alpha, \beta, \gamma, \delta \) have conjugate primitive roots (if \( \neq \epsilon \)).

\[\rightarrow\] case analysis, depending on the emptiness of \( \alpha, \beta, \gamma \)
Conclusion

Theorem

1. It is decidable whether a 2DFT is definable by a 1NFT.
2. It is decidable whether a 2DFT is definable by a 1DFT.
3. Bounded memory is decidable for regular string transductions.

- Complexity: non-elementary upper-bound, PSpace-hard.
Conclusion

Theorem

1. It is decidable whether a 2DFT is definable by a 1NFT.
2. It is decidable whether a 2DFT is definable by a 1DFT.
3. Bounded memory is decidable for regular string transductions.

- Complexity: non-elementary upper-bound, PSpace-hard.

Future Work

- Lower complexity (Shepherdson)
- What about 2NFT (even non functional)?
- Consider other structures: infinite strings, trees
- Variable minimization in streaming string transducers
Classes of Transductions

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

**PTIME**

- \[\text{Choffrut (77)}\]
- \[\text{Weber, Klemm (95)}\]
- \[\text{Beal, Carton, Prieur, Sakarovitch (03)}\]

**DFTs** \(\subset\) \(\subset\) \(\subset\) \(\subset\) \(\subset\)

**fNFTs** \(\subset\) \(\subset\) \(\subset\) \(\subset\) \(\subset\)

**f2NFTs**

\[\equiv\]

**2DFTs**

\[\text{De Souza (13)}\]

**MSOT**

- \[\text{Engelfriet, Hoogeboom (01)}\]
- \[\text{Streaming String Transducers [Alur, Černý, 2010]}\]

**NFTs**

**PTIME**

- \[\text{Schützenberger (75)}\]
- \[\text{Gurari, Ibarra (83)}\]
- \[\text{Beal, Carton, Prieur, Sakarovitch (03)}\]

\[?\]

Decidable

THIS TALK

open
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 
\( \{succ, (lab_a)_{a \in \Sigma}\} \)

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

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

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

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

\[
\phi_{\text{lab}_a}(x) \equiv \text{lab}_a(x)
\]
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: $x := yz$
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: \( x := yz \)

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

\[
\begin{align*}
\alpha | x := \alpha.x & \quad & a | x := x.b \\
& \quad & y := y.c \\
q_0 & \quad & q_0 \\
\rightarrow x & \quad & \rightarrow x.b \\
& \quad & \rightarrow y.c \\
\end{align*}
\]

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

\[ R(T) = a^n\alpha \rightarrow \alpha^{n+1} \]
### 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).
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 $\equiv$ determinitic 2way + $\omega$-regular look-ahead $\equiv$ $\omega$-MSO transductions $\equiv$ $\omega$-SST (Alur,Filiot,Trivedi,12)
It is possible to simulate a \( z \)-motion run with a one-way automaton

1. each state is a triple \((p, r, q)\)
2. the initial state is \((p_0, r_0, q_0)\) with \(q_0 = r_0\)
3. \((p_i, r_i, q_i) \xrightarrow{a} (p_{i+1}, r_{i+1}, q_{i+1})\) iff
   - \(p_i \xrightarrow{a, +1} p_{i+1}\)
   - \(r_{i+1} \xrightarrow{a, -1} r_i\)
   - \(q_i \xrightarrow{a, +1} q_{i+1}\)
4. final states: \((p, p, q)\)