On transducers determinization

Pierre-Alain Reynier

Modelization and Verification team

LIF, Aix-Marseille University & CNRS
Formal methods to improve software

Software systems are complex and ubiquitous

- critical systems $\Rightarrow$ reliability
- widespread $\Rightarrow$ efficiency, scalability

$\Rightarrow$ need for formal methods
Formal methods to improve software

Software systems are complex and ubiquitous

- critical systems $\Rightarrow$ reliability
- widespread $\Rightarrow$ efficiency, scalability

$\Rightarrow$ need for formal methods

Automata-based approaches:

- model checking
- controller synthesis
- performance evaluation
- model optimization

Objective: Improve our theoretical understanding of automata models
### From Languages to Transductions

The table illustrates the comparison between languages and transductions in terms of acceptance and transformation.

<table>
<thead>
<tr>
<th>Languages</th>
<th>Transductions</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Input</em> → {0, 1}</td>
<td><em>Input</em> → <em>Outputs</em></td>
</tr>
<tr>
<td>automata</td>
<td>transducers</td>
</tr>
<tr>
<td>accept inputs</td>
<td>transform inputs</td>
</tr>
</tbody>
</table>

**Applications:**
- Word-to-word transducers: language and speech processing, model-checking infinite state-space systems, reactive systems, verification of web sanitizers.
- Nested-word-to-word transducers: XML transformations, model for recursive programs.
### Applications:

- **Word-to-word transducers:**
  - language and speech processing
  - model-checking infinite state-space systems
  - reactive systems
  - verification of web sanitizers

- **Nested-word-to-word transducers:**
  - XML transformations
  - model for recursive programs
## Simplification of models

### General Problem

Given a (complex) model of a transformation, does there exist an equivalent **simpler** model?

> Natural question:

- minimization of automata
- determinism
- reduce number of registers
- 2way: reduce number of passes
- ...
Overview

1. Introduction
2. Determinization of transducers
3. Register minimization
4. Multi-sequentiality
5. Conclusion
Overview

1. Introduction

2. Determinization of transducers

3. Register minimization

4. Multi-sequentiality

5. Conclusion
Finite-state Transducers

associate output words with transitions of a finite state automaton

Example (A transducer $T$)

![Diagram of a transducer $T$]

Semantics $\llbracket T \rrbracket$: $f : \downarrow \mathbf{w} \downarrow \mapsto a \#_a(\mathbf{w})$, with $\mathbf{w} \in \{a, b\}^*$

Non-determinism: semantics is a relation
Finite-state Transducers

associate output words with transitions of a finite state automaton

Example (A transducer $T$)

$$
\begin{align*}
\begin{array}{c}
\varepsilon \\
\downarrow
\end{array} & \quad \quad \\
\begin{array}{c}
a \\
\downarrow
\end{array} & \quad \quad \\
\begin{array}{c}
b \\
\downarrow
\end{array} & \quad \quad
\end{align*}
$$

Semantics $\llbracket T \rrbracket$: $f : \mathbb{L} \rightarrow a^\#_a(w)$, with $w \in \{a, b\}^*$

Non-determinism: semantics is a relation

A transducer is:

- **functional** if it realizes a function
- **deterministic** if the underlying automaton is deterministic

Classes: DFT, fNFT, NFT
Comparison of classes

DFT = Deterministic Finite-state Transducers
fNFT = Functional Finite-state Transducers
NFT = Non-deterministic Finite-state Transducers

Theorem

\[ DFT \subsetneq fNFT \subsetneq NFT \]

Examples: See blackboard
Determinization Problem

Input: a fNFT $T$
Question: does there exist an equivalent DFT?

Standard technique:

- **subset construction** starting from the set of initial states.
- output **longest common prefix**
- store the **unproduced outputs** in the state

States of the form $\{(p, a), (q, \varepsilon), (s, bb)\}$
An example

\[ \text{dom}(f) = \Sigma^3 \]
\[ f(u) = \text{last}(u)|u| \]
An example

dom(f) = \Sigma^3
f(u) = last(u)|u|

\[
\begin{align*}
\{ (i, \varepsilon) \} & \rightarrow \{ (p_1, a), (q_1, b) \} \\
\{ (p_1, a), (q_1, b) \} & \rightarrow \{ (p_2, aa), (q_2, bb) \} \\
\{ (p_2, aa), (q_2, bb) \} & \rightarrow \{ (p_3, \varepsilon), (q_3, \varepsilon) \} \\
\end{align*}
\]
An example

\[
dom(f) = \Sigma^3
\]

\[
f(u) = \text{last}(u)|u|
\]

Goal: characterize termination of subset construction
Delay between words

**Definition (Longest common prefix)**
Given two words \( u, v \in \Sigma^* \), \( \text{lcp}(u, v) \) denotes the longest common prefix of \( u \) and \( v \).

Example:
\( \text{lcp}(aaa, aab) = aa \)

**Definition (Delay)**
Given two words \( u, v \in \Sigma^* \), we define:

\[
\text{delay}(u, v) = \text{lcp}(u, v)^{-1}.(u, v)
\]

Example:
\( \text{delay}(aaa, aab) = (a, b) \)
Consider some NFT $T$.

**Definition (Twinning Property)**

We say that $T$ satisfies the *twinning property* iff for all situations as depicted on the right, we have:

$$\text{delay}(v_1, w_1) = \text{delay}(v_1 v_2, w_1 w_2)$$
### Lemma

*If a fNFT satisfies the Twinning Property, then the delays computed by the subset construction are bounded.*

### Corollary

\[ \text{Twinning Property} \iff \text{Termination of subset construction.} \]

### Theorem ([WK95])

*Twinning Property can be decided in PTime.*
An example violating the Twinning Property

\[ \text{dom}(f) = \Sigma^+ \]
\[ f(u) = \text{last}(u)^{|u|} \]

After reading an input word \( u \):
- longest common prefix of outputs = \( \varepsilon \)
- subset construction \( \supseteq \{(i_1, a^{|u|}), (i_2, b^{|u|})\} \)

\[ \rightarrow \] The subset construction does not terminate.

The TP is violated: consider synchronised loops around \( i_1 \) and \( i_2 \).
Overview

1 Introduction

2 Determinization of transducers

3 Register minimization

4 Multi-sequentiality

5 Conclusion
Streaming String Transducers [Alur and Cerny, 2010]

**Definition**

Streaming String Transducers (SST for short) are defined as **deterministic** Finite-state automata extended with registers.

Register updates allowed have the following form:

- \( X := u \cdot Y \cdot v \)
- \( X := YZ \)

where \( X, Y, Z \) denote registers and \( u, v \) are words in \( \Sigma^* \).
Streaming String Transducers [Alur and Cerny, 2010]

Definition

Streaming String Transducers (SST for short) are defined as deterministic Finite-state automata extended with registers.

Register updates allowed have the following form:

- \( X := u \cdot Y \cdot v \)
- \( X := YZ \)

where \( X, Y, Z \) denote registers and \( u, v \) are words in \( \Sigma^* \).

Examples:

\[
\begin{align*}
  a, & \quad X := X.a \\
  b, & \quad X := X \\
  \neg w & \quad \mapsto \ a^\# a(w)
\end{align*}
\]
Streaming String Transducers [Alur and Cerny, 2010]

Definition

Streaming String Transducers (SST for short) are defined as deterministic Finite-state automata extended with registers.

Register updates allowed have the following form:

- \( X := u \cdot Y \cdot v \)
- \( X := YZ \)

where \( X, Y, Z \) denote registers and \( u, v \) are words in \( \Sigma^* \).

Examples:

\[
\begin{align*}
a, & \quad \begin{cases} 
X_a := X_a.a \\
X_b := X_b
\end{cases} \\
\vert & \\
b, & \quad \begin{cases} 
X_a := X_a \\
X_b := X_b.b \\
\vert w \vert & \rightarrow a \#(a(w)) b \#(b(w))
\end{cases}
\end{align*}
\]
Another example of SST

Consider the following SST:

\[
\begin{align*}
X_a &:= X_a.a \\
X_b &:= X_b.b
\end{align*}
\]

Which function does it realize?
Another example of SST

Consider the following SST:

\[
  \begin{align*}
    p_a & \xrightarrow{a, \text{upd}} X_a \\
    X_a & \xrightarrow{a, \text{upd}} p_a \\
    p_b & \xrightarrow{b, \text{upd}} X_b \\
    X_b & \xrightarrow{b, \text{upd}} p_b \\
  \end{align*}
\]

where \( \text{upd} : \)

\[
  \begin{align*}
    X_a & := X_a \cdot a \\
    X_b & := X_b \cdot b \\
  \end{align*}
\]

Which function does it realize?

Solution:

\[
  \begin{align*}
    \text{dom}(f) & = \Sigma^* \\
    f(u) & = \text{last}(u) |u|
  \end{align*}
\]
Examples of SST

How to implement these transformations?

- mirror

\[ u = a_1 \ldots a_n \mapsto \tilde{u} = a_n \ldots a_1 \]
Examples of SST

How to implement these transformations?

- **mirror**
  \[ u = a_1 \ldots a_n \mapsto \tilde{u} = a_n \ldots a_1 \]

- **copy**
  \[ u \mapsto uu \]
Examples of SST

How to implement these transformations?

- mirror
  \[ u = a_1 \ldots a_n \mapsto \tilde{u} = a_n \ldots a_1 \]

- copy
  \[ u \mapsto uu \]

- mirror and copy
  \[ u \mapsto \tilde{u}u \]
Examples of SST

How to implement these transformations?

- **mirror**
  \[ u = a_1 \ldots a_n \mapsto \tilde{u} = a_n \ldots a_1 \]

- **copy**
  \[ u \mapsto uu \]

- **mirror and copy**
  \[ u \mapsto \tilde{u}u \]

- **replace**
  \[ u \# v \mapsto v[a \leftarrow u] \]
Examples of SST

How to implement these transformations?

- mirror
  \[ u = a_1 \ldots a_n \mapsto \tilde{u} = a_n \ldots a_1 \]

- copy
  \[ u \mapsto uu \]

- mirror and copy
  \[ u \mapsto \tilde{u}u \]

- replace
  \[ u\#v \mapsto v[a \leftarrow u] \]

- replace2 (\( k \) is fixed)
  \[ u_1\#\ldots\#u_k\#v \mapsto v[a_i \leftarrow u_i] \]

How many registers did you use? Is this number minimal?
Expressiveness results

Comparison of SST and Finite-state Transducers

New class: Two-way Deterministic Finite-state Transducers (2DFT)

\[
\text{DFTs} \subsetneq \text{fNFTs} \subsetneq \text{2DFTs}
\]
Expressiveness results

Comparison of SST and Finite-state Transducers

New class: Two-way Deterministic Finite-state Transducers (2DFT)

\[ DFTs \subseteq fNFTs \subseteq 2DFTs \]

1-register appending SST

\[ X := Xu \]

Long-term objective: Register Minimization for SST

Pierre-Alain Reynier (LIF, team MoVe) On transducers determinization Nov 23, 2017 16 / 25
Expressiveness results

Comparison of SST and Finite-state Transducers

New class: **Two-way** Deterministic Finite-state Transducers (2DFT)

\[
\begin{array}{ccc}
\text{DFTs} & \subsetneq & \text{fNFTs} \\
\equiv & & \equiv \\
1\text{-register appending SST} & \text{appending SST} & \text{2DFTs} \\
\end{array}
\]

\[
\begin{align*}
X & := Xu \\
X & := Yu
\end{align*}
\]

---

Long-term objective: Register Minimization for SST

Pierre-Alain Reynier (LIF, team MoVe)

On transducers determinization  

Nov 23, 2017 16 / 25
Expressiveness results

Comparison of SST and Finite-state Transducers

New class: **Two-way** Deterministic Finite-state Transducers (2DFT)

\[
\begin{align*}
\text{DFTs} & \subseteq \text{fNFTs} & \subseteq \text{2DFTs} \\
\equiv & \quad \equiv & \quad \equiv \\
1\text{-register appending SST} & \quad \text{appending SST} & \quad \text{copyless SST} \\
\uparrow & \quad \uparrow & \quad \uparrow \\
\text{only} & \quad \text{only} & \quad \text{forbid} \\
X := Xu & \quad X := Yu & \quad (X, Y) := (X, X)
\end{align*}
\]

Long-term objective: Register Minimization for SST
Expressiveness results

Comparison of SST and Finite-state Transducers

New class: **Two-way** Deterministic Finite-state Transducers (2DFT)

\[
\begin{align*}
\text{DFTs} & \supsetneq \text{fNFTs} & \supsetneq & \text{2DFTs} \\
1\text{-register appending SST} & \approx & \text{copyless SST} & \subsetneq \text{SST}
\end{align*}
\]

- DFT: only \( X := Xu \)
- fNFT: only \( X := Yu \)
- 2DFT: forbid \((X, Y) := (X, X)\)

Long-term objective: Register Minimization for SST
Expressiveness results

Comparison of SST and Finite-state Transducers

New class: Two-way Deterministic Finite-state Transducers (2DFT)

\[
\begin{align*}
\text{DFTs} & \subsetneq \text{fNFTs} \subsetneq \text{2DFTs} \\
\equiv & \quad \text{1-register appending SST} \\
\equiv & \quad \text{appending SST} \\
\equiv & \quad \text{copyless SST} \\
\equiv & \quad \text{SST}
\end{align*}
\]

\[
\begin{align*}
X := Xu & \quad \uparrow \\
\text{only} & \\
\text{expressiveness}
\end{align*}
\]

\[
\begin{align*}
X := Yu & \quad \uparrow \\
\text{only}
\end{align*}
\]

\[
\begin{align*}
(X, Y) := (X, X) & \quad \uparrow \\
\text{forbid}
\end{align*}
\]

⇒ Long-term objective: Register Minimization for SST
Register minimization for appending SST

\[ \text{dom}(f) = \Sigma^* \]
\[ f(u) = \text{last}(u)|u| \]

\[ \text{upd: } \begin{cases} 
X_a := X_a.a \\
X_b := X_b.b 
\end{cases} \]

\[ \rightarrow \text{can be realized with 2 registers} \]

Can we do better?

\[ \begin{array}{c}
p_a \\
X_a \\
a, \text{ upd} \\
p_b \\
X_b \\
b, \text{ upd} \\
b, \text{ upd} \\
\end{array} \]
Register minimization for appending SST

\[ \text{dom}(f) = \Sigma^* \]
\[ f(u) = \text{last}(u)|u| \]

\[ \text{upd}: \begin{cases} 
X_a := X_a.a \\
X_b := X_b.b 
\end{cases} \]

→ can be realized with 2 registers

Can we do better?
No! 1 register is DFT
Register minimization for appending SST

\[ \text{dom}(f) = \Sigma^* \]
\[ f(u) = \text{last}(u)^{|u|} \]

\[ \text{upd: } \begin{cases} X_a := X_a.a \\ X_b := X_b.b \end{cases} \]

- can be realized with 2 registers

Can we do better?
No! 1 register is DFT

Register Complexity Problem

Input: An appending SST \( T \) and an integer \( k \)
Question: Does there exist a \( k \)-appending SST \( T' \) with \( T \equiv T' \)?
Register complexity using Twinning Property [LICS’16]

Intuition:
2 registers needed if there are 2 runs generating arbitrarily large delays

\( k \) registers needed if there are \( k \) runs generating pairwise arb. large delays
Register complexity using Twinning Property [LICS’16]

Intuition:
2 registers needed if there are 2 runs generating arbitrarily large delays

$k$ registers needed if there are $k$ runs generating pairwise arb. large delays

Contraposition: $T$ satisfies the Twinning Property of order $k$ if:

for every situation like:

\[ \text{there are two runs that remain "close"} \]
Lemma

If $T$ satisfies the TP of order $k$, then from any set of runs on the same input word, one can extract $k$ runs such that every run is "close" from one of these $k$ runs.

Theorem

A fNFT is definable by a $k$-appending SST iff it satisfies the TP of order $k$.

Theorem

Given a fNFT $T$ and $k$ (in unary), deciding whether $T$ satisfies the TP of order $k$ is PSpace-complete.
Register complexity using Twinning Property \([\text{LICS'16}]\)

An example: how many registers for the following function?

\[
f : u_1 \# u_2 \mapsto \text{last}(u_1)^{|u_1|} \# \text{last}(u_2)^{|u_2|}
\]

\[
\begin{array}{c}
\sigma | a \\
\sigma | a \\
\sigma | b \\
\sigma | b \\
\end{array}
\begin{array}{c}
a | a \\
# | # \\
b | b \\
b | b \\
\end{array}
\begin{array}{c}
\sigma | a \\
\sigma | a \\
\sigma | a \\
\sigma | b \\
\end{array}
\begin{array}{c}
a | a \\
# | # \\
# | # \\
# | # \\
\end{array}
\begin{array}{c}
\sigma | a \\
\sigma | a \\
\sigma | b \\
\sigma | b \\
\end{array}
\begin{array}{c}
# | # \\
\# | # \\
\# | # \\
\# | # \\
\end{array}
\begin{array}{c}
\sigma | a \\
\sigma | a \\
\sigma | b \\
\sigma | b \\
\end{array}
\begin{array}{c}
b | b \\
\# | # \\
b | b \\
b | b \\
\end{array}
\begin{array}{c}
\sigma | a \\
\sigma | a \\
\sigma | b \\
\sigma | b \\
\end{array}
\begin{array}{c}
\# | # \\
\# | # \\
\# | # \\
\# | # \\
\end{array}
\begin{array}{c}
\sigma | a \\
\sigma | a \\
\sigma | b \\
\sigma | b \\
\end{array}
\begin{array}{c}
b | b \\
\# | # \\
b | b \\
b | b \\
\end{array}
\begin{array}{c}
\sigma | a \\
\sigma | a \\
\sigma | b \\
\sigma | b \\
\end{array}
\begin{array}{c}
\# | # \\
\# | # \\
\# | # \\
\# | # \\
\end{array}
\begin{array}{c}
\sigma | a \\
\sigma | a \\
\sigma | b \\
\sigma | b \\
\end{array}
\begin{array}{c}
b | b \\
\# | # \\
b | b \\
b | b \\
\end{array}
\end{array}
Register complexity using Twinning Property \[\text{[LICS'16]}\]

An example: how many registers for the following function?

\[ f : u_1 \# u_2 \mapsto \text{last}(u_1)^{u_1} \# \text{last}(u_2)^{u_2} \]

Only 2 registers!
Register complexity using Twinning Property [LICS’16]

An example: how many registers for the following function?

\[ f : u_1 \# u_2 \mapsto \text{last}(u_1)^{|u_1|} \# \text{last}(u_2)^{|u_2|} \]

\[ X_a := X_b \cdot \# \]
\[ X_b := X_a \cdot \# \]
Overview

1. Introduction
2. Determinization of transducers
3. Register minimization
4. Multi-sequentiality
5. Conclusion
Multi-sequential functions [CS86]

Definition

A function $f$ is multi-sequential if there exists a finite number of DFTs $T_1, \ldots, T_n$ such that $f = \bigcup_i [T_i]$.

Example: $f(u) = \text{last}(u)|u|$
Multi-sequential functions [CS86]

**Definition**

A function $f$ is multi-sequential if there exists a finite number of DFTs $T_1, \ldots, T_n$ such that $f = \bigcup_i [T_i]$.

Example: $f(u) = \text{last}(u)|u|$.

**Theorem**

$DFT \subsetneq \text{multi-seq} \subsetneq fNFT$

Examples: see blackboard.
Multi-sequential functions [CS86]

**Definition**
A function $f$ is **multi-sequential** if there exists a finite number of DFTs $T_1, \ldots, T_n$ such that $f = \bigcup_i [T_i]$

Example: $f(u) = \text{last}(u) |u|$

**Theorem**

$DFT \subset \text{multi-seq} \subset fNFT$

Examples: see blackboard

**$k$-sequentiality Problem**

Input: A $fNFT T$ and an integer $k$

Question: Does there exist $k$ DFT $T_1, \ldots, T_k$ such that $[T] = \bigcup_i [T_i]$?
Multi-sequential functions as SST [FoSSaCS'17]

Theorem

A function can be realized as a union of $k$ DFT iff it can be realized by a SST with $k$ registers and updates of the form $X := Xu$.

→ Solving the $k$-sequentiality problem amounts to solve register minimization in this class.
Multi-sequential functions as SST \cite{FoSSaCS'17}

**Theorem**

A function can be realized as a union of \( k \) DFT iff it can be realized by a SST with \( k \) registers and updates of the form \( X := Xu \).

⇒ Solving the \( k \)-sequentiality problem amounts to solve register minimization in this class.

⇒ introduction of a “Branching” Twinning Property of order \( k \) (input words can be different between runs)
Multi-sequential functions as SST [FoSSaCS’17]

**Theorem**

A function can be realized as a union of \( k \) DFT iff it can be realized by a SST with \( k \) registers and updates of the form \( X := Xu \).

- Solving the \( k \)-sequentiality problem amounts to solve register minimization in this class.

- Introduction of a “Branching” Twinning Property of order \( k \) (input words can be different between runs)

**Theorem**

A \( fNFT \) is definable by a union of \( k \) DFT iff it satisfies the Branching TP of order \( k \).
### Theorem

A function can be realized as a union of \( k \) DFT iff it can be realized by a SST with \( k \) registers and updates of the form \( X := Xu \).

→ Solving the \( k \)-sequentiality problem amounts to solve register minimization in this class.

→ introduction of a “Branching” Twinning Property of order \( k \) (input words can be different between runs)

### Theorem

A fNFT is definable by a union of \( k \) DFT iff it satisfies the Branching TP of order \( k \).

### Theorem

Given a fNFT \( T \) and \( k \) (in unary), deciding whether \( T \) satisfies the Branching TP of order \( k \) is PSpace-complete.
Example

Consider this function:

\[ f : u_1 \# u_2 \mapsto \]
\[ \text{last}(u_1)^{|u_1|} \# \text{last}(u_2)^{|u_2|} \]

Is it 2-sequential?
Is it 3-sequential?
Is it 4-sequential?
Example

Consider this function:

\[ f : u_1 \# u_2 \mapsto \text{last}(u_1)^{|u_1|} \# \text{last}(u_2)^{|u_2|} \]

Is it 2-sequential?
Is it 3-sequential?
Is it 4-sequential?

Minimum = 4
Overview

1. Introduction

2. Determinization of transducers

3. Register minimization

4. Multi-sequentiality

5. Conclusion
Summary

Some “old” results:

- Finite-state Transducers
- Determinization not always possible for transducers
- Twinning Property to characterize determinizability

Recent applications to:

- Streaming String Transducer
- Register minimization
- Multi-sequential functions
Summary

Some “old” results:
- Finite-state Transducers
- Determinization not always possible for transducers
- Twinning Property to characterize determinizability

Recent applications to:
- Streaming String Transducer
- Register minimization
- Multi-sequential functions

I did not present...
- other decidability results (2Way $\sim$ 1Way)
- logical characterization
- algebraic characterization
Some perspectives

Extend register minimization to larger classes of SST
  - ongoing work on updates $X := uYv$
  - algebraic presentation, canonical object

Specification languages for transformations

Alternative semantics to break undecidability/high complexity
The Modelization and Verification team

Research topics:

• Language theory
  transducers, higher-order languages, weighted automata

• Algorithmic verification
  infinite state systems such as Petri nets, timed systems

• Security applications
  workflows, access control, quantitative information flow
The Modelization and Verification team
Thanks!