Towards Register Minimisation of Streaming String Transducers

Pierre-Alain Reynier

LIS, Aix-Marseille Université & CNRS
Transducers

Automata accept objects / Transducers transform objects

A transduction is a function (or even a relation) from words to words

→ In this talk, we focus on functions

Examples:

→ **ERASE**: “Oxford” $\mapsto$ “xfrd”
→ **LAST**: “Oxford” $\mapsto$ “ddddddd”
→ **REVERSE**: “Oxford” $\mapsto$ “drofxO”
→ **COPY**: “Oxford” $\mapsto$ “OxfordOxford”
→ **REPLACE**: “Oxford#I love $1” $\mapsto$ “I love Oxford”
→ **SORT**: “Oxford” $\mapsto$ “dfoOrx”
Transducers

Some applications:

- language and speech processing
- model-checking infinite state-space systems
- verification of web sanitizers
- string pattern matching
- XML transformations (nested word)
- model for recursive programs (nested word)
(One/Two-way) finite state transducers

Example (A transducer $T$)

Semantics $\sem{T}$: $\text{ERASE} : \vdash w \dashv \leftrightarrow a^{\#_a(w)}$, with $w \in \{a, b\}^*$

Non-determinism: semantics is a relation
(One/Two-way) finite state transducers

Example (A transducer $T$)

Semantics $\llbracket T \rrbracket$: ERASE: $\vdash w \dashv \iff a \# a(w)$, 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: det1W, fun1W, 1W

Too low expressive power (Reverse, Copy, Replace, Sort)
(One/Two-way) finite state transducers

Example (A transducer $T$)

Semantics $\llbracket T \rrbracket$: $\text{SORT} : \vdash w \dashv \mapsto a^\#_a(w) b^\#_b(w)$, 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: det1W, fun1W, 1W, det2W, fun2W, 2W
Regular Word Functions

\[ \text{fun2W} = \text{det2W} \]
Regular Word Functions

MSO-definable Transducers (à la Courcelle)

fun2W = det2W

[EH01]

[EH01]
Regular Word Functions

MSO-definable Transducers

[EH01] fun2W = det2W

[AC10] copyless Streaming String Transducers (SST)
Regular Word Functions

- MSO-definable Transducers
  - [EH01]
  - $\text{fun}2W = \text{det}2W$

- copyless Streaming String Transducers (SST)
  - [AC10]

- Regular Functions Expressions
  - [BR18]
  - [AFR14]
Regular Word Functions

- closed under composition
- regular languages are preserved by inverse image
- functionality and equivalence are decidable
Streaming String Transducers [AC10]

$1W$ deterministic autom.
+ registers

Register updates:
- $X := u.Y.v$
- $X := Y.Z$

$X, Y, Z$: registers
$u, v$: words in $\Sigma^*$

Expression results:
$\det_{1W} \equiv$ 1-register appending SST $X := X.a$
$\fun_{1W} \equiv$ appending SST $X := Y.a$
$\fun_{2W} \equiv$ copyless SST $(X, Y) := (X, X)$ is forbidden

$\vdash w \vdash \mapsto a \#_a(w) \ b \#_b(w)$

\[
\begin{align*}
  a & \begin{cases}
    X_a := X_a.a \\
    X_b := X_b
  \end{cases} \\
  b & \begin{cases}
    X_a := X_a \\
    X_b := X_b.b
  \end{cases}
\end{align*}
\]
Streaming String Transducers \cite{AC10}

1W deterministic autom.
+ registers

Register updates:
- $X := u.Y.v$
- $X := Y.Z$

$X, Y, Z$: registers
$u, v$: words in $\Sigma^*$

Expressiveness results:
- $\text{det1W} \equiv 1$-register appending SST

```
¬w¬ ⊢ a#_a(w) b#_b(w)
```

```
a \{ X_a := X_a.a
 X_b := X_b
\}

b \{ X_a := X_a
 X_b := X_b.b
\}
```

```
X := X.a
```
Streaming String Transducers [AC10]

1W deterministic autom. + registers

Register updates:
- $X := u \cdot Y \cdot v$
- $X := Y \cdot Z$

$X, Y, Z$: registers
$u, v$: words in $\Sigma^*$

Expressiveness results:
- $\text{det}1W \equiv 1$-register appending SST
- $\text{fun}1W \equiv$ appending SST

$\vdash w \vdash \mapsto a^\#_a(w) b^\#_b(w)$
Streaming String Transducers [AC10]

1W deterministic autom.  
+ registers

Register updates:
- \(X := u \cdot Y \cdot v\)
- \(X := Y \cdot Z\)

\(X, Y, Z\): registers
\(u, v\): words in \(\Sigma^*\)

Expressiveness results:
- \(\text{det1W} \equiv 1\)-register appending SST
- \(\text{fun1W} \equiv\) appending SST
- \(\text{fun2W} \equiv\) copyless SST

\(\vdash w \vdash \rightarrow a \#_a(w) \ b \#_b(w)\)

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

\(X := X \cdot a\)
\(X := Y \cdot a\)

\((X, Y) := (X, X)\) is forbidden
Examples of SST

\[ \sigma | X := \sigma.X \]

\[ \sigma | X := X.\sigma \]

\[ a | up \]

\[ b | up \]

\[ b | up \]

\[ X_a := X_a.a \]

\[ X_b := X_b.b \]

\[ \sigma \neq \# \]

\[ X := X.\sigma \]

\[ Y := \varepsilon \]

\[ \sigma \neq $1 \]

\[ X := X \]

\[ Y := Y.\sigma \]

\[ \# \]

\[ $1 \]

\[ X := X \]

\[ Y := Y.X \]
Register Minimisation Problem for SST

Motivations: Streaming and simplification of models

- minimisation/determinisation of automata
- normal form \( \sim \) learning
- 2way: reduce number of passes

Register Minimisation Problem for class \( S \) of SST

Input: \( T \in S \) and \( k \in \mathbb{N} \)

Question: Does there exist \( T' \in S \) with \( k \) registers s.t. \( T \equiv T' \)?

Related works

- [AR13] Additive Cost Register Automata
  \( X := Y + c, \ c \in \mathbb{Z} \)
- [BGMP16] concatenation-free funNSST
  \( X := uYv \)
Classes of Functions

Regular functions  \[
det2W=\text{copyless} \quad \text{SST}=\text{MSOT}
\]

Reverse  Copy
Classes of Functions

Regular functions
\[ \text{det2W = copyless SST = MSOT} \]

Rational functions
\[ \text{fun1W = appending SST} \]
\[ X := Y \cdot u \]

Reverse
Copy

Last
Classes of Functions

Regular functions
\[ \text{det}2W = \text{copyless} \quad \text{SST} = \text{MSOT} \]

Rational functions
\[ \text{fun}1W = \text{appending SST} \quad X := Y.u \]

Sequential functions
\[ \text{det}1W = 1\text{-app.SST} \]

Erase
Last

Reverse
Copy

Pierre-Alain Reynier (LIS, AMU & CNRS)  Towards Register Minimisation of SST  Oxford, Feb 22, 2018  9 / 25
Classes of Functions

Regular functions \( \text{det2W=} \text{copyless} \quad \text{SST=} \text{MSOT} \)

- Rational functions
  \( \text{fun1W=} \text{appending} \quad \text{SST} \quad X:=Y.u \)

- Sequential functions
  \( \text{det1W=} \text{1-app.SST} \)
  \( \text{ERASE} \)

- Multi-seq. functions
  \( \text{X:=X.u} \)
  \( \text{LAST} \)

- Reverse

- Copy
In this talk

- Rational functions ($X := Y.u$)
  ➞ [LICS16] with L. Daviaud and J.M. Talbot

- Multi-sequential functions ($X := X.u$)
  ➞ [FoSSaCS17] with L. Daviaud, I. Jecker and D. Villevalois
Overview

1. Introduction

2. Rational functions \((X := Y \cdot u)\)

3. Multi-sequential functions \((X := X \cdot u)\)

4. Conclusion
Overview

1. Introduction

2. Rational functions \((X := Y \cdot u)\)

3. Multi-sequential functions \((X := X \cdot u)\)

4. Conclusion
Rational functions and appending SST

Appending SST: only updates $X := Y \cdot u$

Facts:
- appending SST $=$ fun1W
- appending SST $\leadsto$ fun1W is polynomial (guess the register)
- appending SST with 1 register $=$ det1W

Register minimisation for appending SST

**Input:** an appending SST $T$ and $k \in \mathbb{N}$

**Question:** does there exist an app. SST $T'$ with $k$ registers s.t. $T \equiv T'$?

$\Rightarrow$ for $k = 1$, our problem is the det1W-definability of fun1W
From rational functions to sequential ones

Sequentiality Problem [Choffrut77]

**Input:** a function $T$

**Question:** does there exist an equivalent deterministic function $det T$?

Standard technique:

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

Configurations of the form $\{(p, a), (q, \varepsilon), (s, bb)\}$
From rational functions to sequential ones

Sequentiality Problem [Choffrut77]

**Input:** a fun1W $T$

**Question:** does there exist an equivalent det1W?

Standard technique:

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

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

**Issue:** termination (bound the size of unproduced outputs)
An example

\texttt{LAST} on $\Sigma^3$
An example

**LAST** on $\Sigma^3$

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

\[
\begin{align*}
\sigma|a & \quad p_1 \\
\sigma|b & \quad q_1 \\
& \quad p_2 \\
& \quad q_2 \\
& \quad p_3 \\
& \quad q_3
\end{align*}
\]
Twinning Property [Choffrut77]

We define:

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

Example:
\[ \text{lcp}(aaa, aab) = aa \]
\[ \text{delay}(aaa, aab) = (a, b) \]

For all situations like:
\[ \text{we have delay}(w_0, w_1) = \text{delay}(w_0 w'_0, w_1 w'_1) \]
Twinning Property [Choffrut77]

We define:

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

Example:
\[
\text{lcp}(aaa, aab) = aa
\]
\[
\text{delay}(aaa, aab) = (a, b)
\]

For all situations like:

\[
\begin{align*}
u | w_0 & \quad \longrightarrow \quad v | w_0' \\
v | w_1 & \quad \longrightarrow \quad u | w_1'
\end{align*}
\]

we have \( \text{delay}(w_0, w_1) = \text{delay}(w_0 w_0', w_1 w_1') \)

\( T \models \text{Twinning Property} \implies \forall (p, x) \in \text{subset constr.}, |x| \leq n^2 M \)

Theorem ([Choffrut77])

\( T \models \text{Twinning Property} \iff \text{There exists an equivalent det1W} \)
Twinning Property [Choffrut77]

We define:

$$\text{delay}(u, v) = \text{lcp}(u, v)^{-1} \cdot (u, v)$$

Example:

$$\text{lcp}(aaa, aab) = aa$$
$$\text{delay}(aaa, aab) = (a, b)$$

For all situations like:

For all situations like:

$$\text{delay}(w_0, w_1) = \text{delay}(w_0 w'_0, w_1 w'_1)$$

we have

\[ T \models \text{Twinning Property} \implies \forall (p, x) \in \text{subset constr.}, |x| \leq n^2 M \]

Theorem ([Choffrut77])

\[ T \models \text{Twinning Property} \iff \text{There exists an equivalent det1W} \]

Theorem ([WK95])

Twinning Property can be decided in PTime.
Register minimisation using Twinning Property

Our objective: Characterize when a \texttt{fun1W} can be expressed by an appending SST with $k$ registers.

Twinning property characterizes the fact that runs (on the same input) remain close.

Intuition:
2 reg. needed if there are 2 runs with arbitrarily large delays

$k + 1$ reg. needed if there are $k + 1$ runs with \texttt{pairwise} arb. large delays

$k$ registers are sufficient if for every $k + 1$ runs, 2 of them remain close
Register minimisation using Twinning Property

Our objective: Characterize when a fun1W can be expressed by an appending SST with \( k \) registers.

Twinning property characterizes the fact that runs (on the same input) remain close.

Intuition:

- 2 reg. needed if there are 2 runs with arbitrarily large delays
- \( k + 1 \) reg. needed if there are \( k + 1 \) runs with pairwise arb. large delays
- \( k \) registers are sufficient if for every \( k + 1 \) runs, 2 of them remain close

For every \( k + 1 \) runs, 2 of them remain close
Twinning Property of order $k$

For all situations like:

$k$ synchronised loops

there are two runs $0 \leq i < j \leq k$ s.t. for every loop $\ell$,

we have $\text{delay}(w_{1,i} \ldots w_{\ell,i}, w_{1,j} \ldots w_{\ell,j}) = \text{delay}(w_{1,i} \ldots w_{\ell,i} w'_{i}, w_{1,j} \ldots w_{\ell,j} w'_{j})$
Lemma

If a fun1W 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" to one of these $k$ runs.

"close": $(p, x)$ with $|x| \leq n^{k+1} M$
Register minimisation using Twinning Property

Lemma

If a fun1W 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" to one of these $k$ runs.

"close": $(p, x)$ with $|x| \leq n^{k+1}M$

Theorem

- A fun1W is definable by a $k$-app. SST iff it satisfies the TP of order $k$
- TP of order $k$ can be decided in PSpace ($k$ given in unary)
Register minimisation using Twinning Property

Lemma

If a fun1W 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" to one of these $k$ runs.

"close": $(p, x)$ with $|x| \leq n^{k+1} M$

Theorem

- A fun1W is definable by a $k$-app. SST iff it satisfies the TP of order $k$
- TP of order $k$ can be decided in PSpace ($k$ given in unary)

Corollary

The register minimisation problem for appending SST is PSpace-complete.
Example

How many registers for the following function?

\[ \text{LAST}^2 : u_1 \# u_2 \mapsto \text{LAST}(u_1) \# \text{LAST}(u_2) \]
Example

How many registers for the following function?

$$\text{LAST}^2 : u_1 \# u_2 \mapsto \text{LAST}(u_1) \# \text{LAST}(u_2)$$

Only 2 registers!
Example

\[ \text{LAST}^2 : \ u_1 \# u_2 \mapsto \text{LAST}(u_1) \# \text{LAST}(u_2) \]

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

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

1. Introduction

2. Rational functions \((X := Y \cdot u)\)

3. Multi-sequential functions \((X := X \cdot u)\)

4. Conclusion
Multi-sequential functions

Definition ([CS86])
Multi-sequential functions are defined as functions that can be realized as finite union of sequential transducers.

→ allows a parallel evaluation in a streaming scenario

Examples:
- \textsc{Last} on \( \Sigma = \{a, b\} \) is multi-sequential: split \( \Sigma^+ \) as \( \Sigma^* a \cup \Sigma^* b \)
Multi-sequential functions

Definition ([CS86])

Multi-sequential functions are defined as functions that can be realized as finite union of sequential transducers.

→ allows a parallel evaluation in a streaming scenario

Examples:

- `Last` on $\Sigma = \{a, b\}$ is multi-sequential: split $\Sigma^+$ as $\Sigma^*a \cup \Sigma^*b$

- `Last^2 : u_1 \# u_2 \mapsto Last(u_1) \# Last(u_2)` is multi-sequential: split the domain according to `last(u_1), last(u_2) \in \{a, b\}`
Multi-sequential functions

Definition ([CS86])

Multi-sequential functions are defined as functions that can be realized as finite union of sequential transducers.

→ allows a parallel evaluation in a streaming scenario

Examples:

- $\text{Last}$ on $\Sigma = \{a, b\}$ is multi-sequential: split $\Sigma^+$ as $\Sigma^*a \cup \Sigma^*b$
- $\text{Last}^2 : u_1 \# u_2 \mapsto \text{Last}(u_1) \# \text{Last}(u_2)$ is multi-sequential:
  split the domain according to $\text{last}(u_1), \text{last}(u_2) \in \{a, b\}$
- $\text{Last}^* : u_1 \# \ldots \# u_n \mapsto \text{Last}(u_1) \# \ldots \# \text{Last}(u_n)$ is not multi-seq.
Multi-sequential functions

Definition ([CS86])

Multi-sequential functions are defined as functions that can be realized as finite union of sequential transducers.

Definition (Appending SST with independent registers)

Only updates \( X := Xu \): "No communication between threads"
Multi-sequential functions

**Definition ([CS86])**
Multi-sequential functions are defined as functions that can be realized as finite union of sequential transducers.

**Definition (Appending SST with independent registers)**
Only updates $X := Xu$: ”No communication between threads”

Observations:
- Multi-sequential functions $\equiv$ app. SST with independent registers
- size of the union $\equiv$ number of registers

→ Register minimisation in this class $\equiv$ Minimisation of size of the union
Example

\[ \text{LAST}^2 : u_1 \# u_2 \mapsto \text{LAST}(u_1) \# \text{LAST}(u_2) \]

\[
\begin{array}{c}
\sigma | b \\
\sigma | a \\
\sigma | a \\
\sigma | b \\
\sigma | b \\
\end{array}
\]

\[
\begin{array}{c}
\sigma | b \\
\sigma | b \\
\# | # \\
\sigma | a \\
\sigma | a \\
\end{array}
\]

\[\Rightarrow \text{Requires 4 independent registers}\]

Registers cannot be reset!
Branching twinning property of order $k$

For all situations like:

$k$ not synchronised loops

$k + 1$ runs

there are two runs $0 \leq i < j \leq k$ s.t. for every loop $\ell$ with same input words, we have

$$\text{delay}(w_{1,i} \ldots w_{\ell,i}, w_{1,j} \ldots w_{\ell,j}) = \text{delay}(w_{1,i} \ldots w_{\ell,i}w'_{\ell,i}, w_{1,j} \ldots w_{\ell,j}w'_{\ell,j})$$
Branching twinning property of order $k$

Tree representation of input words:
Branching twinning property of order $k$

**Theorem**

- A $\text{fun1W}$ is definable by a $k$-app. SST with independent registers iff it satisfies the BTP of order $k$.
- The BTP of order $k$ is decidable in $\text{PSpace}$ ($k$ in unary).
Branching twinning property of order $k$

**Theorem**

- A $\text{fun1W}$ is definable by a $k$-app. SST with independent registers iff it satisfies the BTP of order $k$.
- The BTP of order $k$ is decidable in PSpace ($k$ in unary).

**Theorem**

*The register minimisation problem for appending SST with independent registers is PSpace-complete.*
Overview

1. Introduction

2. Rational functions \((X := Y \cdot u)\)

3. Multi-sequential functions \((X := X \cdot u)\)

4. Conclusion
Summary

Regular functions

Regular functions
det2W=copyless SST=MSOT

Rational functions
fun1W=appending SST
X:=Y.u

Rational functions

det1W
TP

Multi-seq. functions

REVERSE
COPY

X:=Y.u
Summary

**Regular functions**

- det2W = copyless SST = MSOT

**Rational functions**

- fun1W = appending SST

- X := Y \cdot u

**Multi-seq. functions**

- 2-app. SST
- TP of order 2

- det1W
- TP

**Reverse**

**Copy**
Regular functions  
\[ \text{det2W=} \text{copyless SST=} \text{MSOT} \]

Rational functions  
\[ \text{fun1W=} \text{appending SST} \quad X:=Y.u \]

k-app. SST: TP of order \( k \)

2-app. SST  
TP of order 2

\[ \text{det1W} \quad \text{TP} \]
Summary

Regular functions

\[ \text{det}2W = \text{copyless SST} = \text{MSOT} \]

Rational functions

\[ \text{fun}1W = \text{appending SST} \]

\[ X := Y\cdot u \]

k-app. SST: TP of order \( k \)

2-app. SST

TP of order 2

det1W

TP

2-seq. BTP of order 2

REVERSE

COPY
Summary

### Regular functions

- **det2W** = copyless
- **SST** = **MSOT**

---

### Rational functions

- **fun1W** = appending
- **SST**

#### k-app. SST: TP of order $k$

- **2-app. SST**
  - TP of order 2

---

**REVERSE**

**COPY**

- **det1W**
  - TP

- **2-seq. BTP of order 2**
- **k-seq. BTP of order k**
Summary

Regular functions \( \text{det2W=} \text{copyless SST=} \text{MSOT} \)

Rational functions
fun1W=appending SST \( \ X:=Y.u \)

k-app. SST: TP of order \( k \)

2-app. SST
TP of order 2

Multi-seq. functions

Multi-seq. functions
2-seq. BTP of order 2

k-seq. BTP of order \( k \)

Reverse
Copy
Alternative characterizations:

- bounded variation property
- Lipschitz property
Alternative characterizations:
- bounded variation property
- Lipschitz property

Functional $\sim$ finite-valued
Alternative characterizations:

- bounded variation property
- Lipschitz property

Functional $\sim$ finite-valued

Extension to ”weak” weighted automata on semigroups:

- set semantics
- infinitary semigroup ($\alpha \beta \gamma \neq \beta \implies |\{\alpha^n \beta \gamma^n | n \in \mathbb{N}\}| = +\infty$)
- finitely generated semigroup
Shift from rational to regular functions

- deal with both prepending and appending: \( X := u.Y.v \) (on-going)
- deal with concatenation of registers

Weighted automata: replace set semantics with other aggregations

Extensions to infinite words, nested words
Perspectives

Shift from rational to regular functions
→ deal with both prepending and appending: $X := u.Y.v$ (on-going)
→ deal with concatenation of registers

Weighted automata: replace set semantics with other aggregations

Extensions to infinite words, nested words

Thanks!
Classes of Transductions

Regular functions
\[ \text{det}2W = \text{copyless SST} = \text{MSOT} \]

**Copy**

**Reverse**

Rational functions
\[ \text{fun}1W = \text{appending SST} \]

Rational relations
\[ 1W = \text{appending NSST} \]

**Subword**
\[ u \mapsto \{ u' \mid u' \preceq u \} \]

**Kleene Star**
\[ u \mapsto \{ u^* \} \]

\[ 2W = \text{NMSOT} \]

**Subwords**
\[ u \mapsto \{ u' u' \mid u' \preceq u \} \]
Classes of Transductions

Rational functions
\( \text{fun1W=appending SST} \)
\( (X:=Y.u) \)

\( \text{LAST} \)

Regular functions
\( \text{det2W=copyless SST} \)
\( =\text{MSOT} \)

\( \text{COPY} \)

\( \text{REVERSE} \)
Classes of Transductions

Regular functions
- $\text{det}2W = \text{copyless SST} = \text{MSOT}$
- $\text{Copy}$
- $\text{Reverse}$

Rational functions
- $\text{fun}1W = \text{appending SST} = \text{LAST}$
- $\text{Subword} \ u \mapsto \{ u' \mid u' \preceq u \}$

Rational relations
- $1W = \text{appending NSST}$
- $\text{Subword} \ u \mapsto \{ u' \mid u' \preceq u \}$
Classes of Transductions

Kleene Star $u \mapsto u^*$

2W

Rational relations
$1W =$ appending NSST

Subword $u \mapsto \{u' | u' \leq u\}$

Rational functions
fun1W = appending SST
$(X:=Y.u)$

Last

Regular functions
det2W = copyless SST
= MSOT

Copy

Reverse
## Classes of Transductions

<table>
<thead>
<tr>
<th>Classes</th>
<th>2W</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kleene Star</strong> $u \mapsto u^*$</td>
<td></td>
</tr>
<tr>
<td><strong>Rational relations</strong></td>
<td>1W=appending NSST</td>
</tr>
<tr>
<td><strong>Subword</strong> $u \mapsto { u' \mid u' \preceq u }$</td>
<td></td>
</tr>
<tr>
<td><strong>Rational functions</strong></td>
<td>fun1W=appending SST</td>
</tr>
<tr>
<td>$X:=Y.u$</td>
<td></td>
</tr>
<tr>
<td><strong>Last</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Regular functions</strong></td>
<td>det2W=copyless SST</td>
</tr>
<tr>
<td>$u \mapsto { u'u' \mid u' \preceq u }$</td>
<td></td>
</tr>
<tr>
<td><strong>Copy</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Reverse</strong></td>
<td></td>
</tr>
</tbody>
</table>
### Alternative characterizations

\[ f : \Sigma^* \rightarrow \Gamma^* \]

<table>
<thead>
<tr>
<th></th>
<th>bounded variation</th>
<th>Lipschitz property</th>
</tr>
</thead>
<tbody>
<tr>
<td>det1W</td>
<td>$\forall n : \exists N : \forall u, v \in \text{dom}(f), \quad d(u, v) \leq n \Rightarrow d(f(u), f(v)) \leq N$</td>
<td>$\exists L : \forall u, v \in \text{dom}(f), \quad d(f(u), f(v)) \leq L.(d(u, v) + 1)$</td>
</tr>
<tr>
<td>$k$ registers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k$ independent registers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Alternative characterizations

\[ f : \Sigma^* \mapsto \Gamma^* \]

<table>
<thead>
<tr>
<th></th>
<th>bounded variation</th>
<th>Lipschitz property</th>
</tr>
</thead>
<tbody>
<tr>
<td>det1W</td>
<td>( \forall n \exists N \forall u, v \in \text{dom}(f), d(u, v) \leq n \Rightarrow d(f(u), f(v)) \leq N )</td>
<td>( \exists L \forall u, v \in \text{dom}(f), d(f(u), f(v)) \leq L.(d(u, v) + 1) )</td>
</tr>
<tr>
<td>( k ) registers</td>
<td>( \forall n \exists N \forall u_0 \ldots u_k \in \text{dom}(f), ) ( (\forall i \neq j, d(u_i, u_j) \leq n) ) ( \Rightarrow \exists i \neq j . d(f(u_i), f(u_j)) \leq N )</td>
<td>?</td>
</tr>
<tr>
<td>( k ) independent registers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Alternative characterizations

\[ f : \Sigma^* \rightarrow \Gamma^* \]

<table>
<thead>
<tr>
<th></th>
<th>bounded variation</th>
<th>Lipschitz property</th>
</tr>
</thead>
<tbody>
<tr>
<td>det1W</td>
<td>( \forall n \exists N \forall u, v \in \text{dom}(f), ) ( d(u, v) \leq n \Rightarrow d(f(u), f(v)) \leq N )</td>
<td>( \exists L \forall u, v \in \text{dom}(f), ) ( d(f(u), f(v)) \leq L.(d(u, v) + 1) )</td>
</tr>
<tr>
<td>(k) registers</td>
<td>( \forall n \exists N \forall u_0 \ldots u_k \in \text{dom}(f), ) ( (\forall i \neq j, d(u_i, u_j) \leq n) \Rightarrow \exists i \neq j ). ( d(f(u_i), f(u_j)) \leq N )</td>
<td>?</td>
</tr>
<tr>
<td>(k) independent registers</td>
<td>?</td>
<td>( \exists L \forall u_0 \ldots u_k \in \text{dom}(f), ) ( \exists i \neq j \text{ s.t.} ) ( d(f(u_i), f(u_j)) \leq L.(d(u_i, u_j) + 1) )</td>
</tr>
</tbody>
</table>