

## Comparison-free polyregular functions

---

NGUYỄN Lê Thành Dũng (a.k.a. Tito) — [nltd@nguyentito.eu](mailto:nltd@nguyentito.eu)  
Laboratoire d'informatique de Paris Nord, Villetaneuse, France

Camille Noûs — <https://www.cogitamus.fr/camilleen.html>

Pierre PRADIC — [pierre.pradic@cs.ox.ac.uk](mailto:pierre.pradic@cs.ox.ac.uk)  
University of Oxford, United Kingdom

ICALP 2021 — track B

## **Introduction: string-to-string transducers**

---

# Deterministic two-way transducers (2DFT)

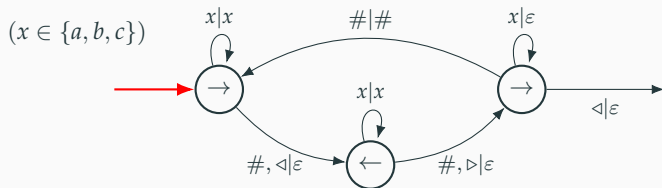
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:



# Deterministic two-way transducers (2DFT)

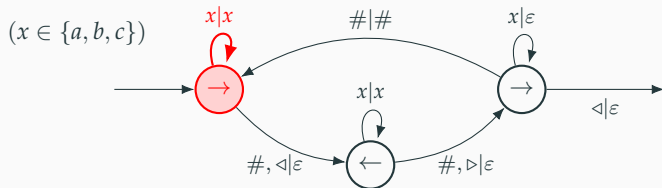
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

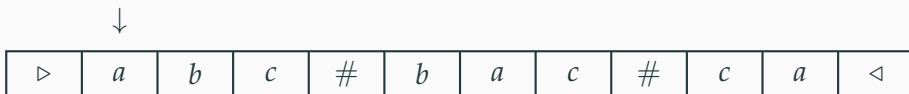
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:



# Deterministic two-way transducers (2DFT)

Topic of this talk: certain basic computation models for string-to-string functions

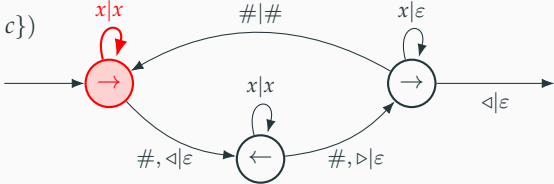
## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

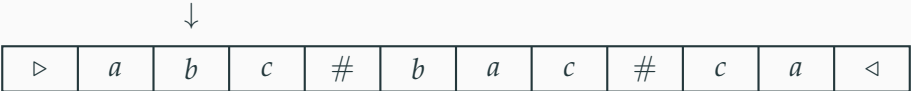
Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$

$(x \in \{a, b, c\})$



Output:  
a



# Deterministic two-way transducers (2DFT)

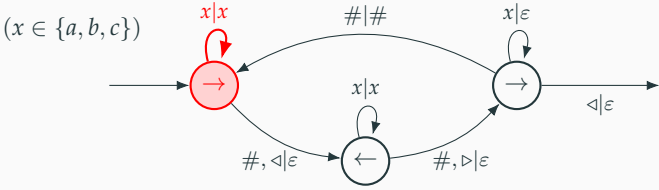
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

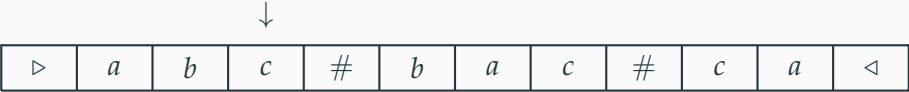
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*ab*



# Deterministic two-way transducers (2DFT)

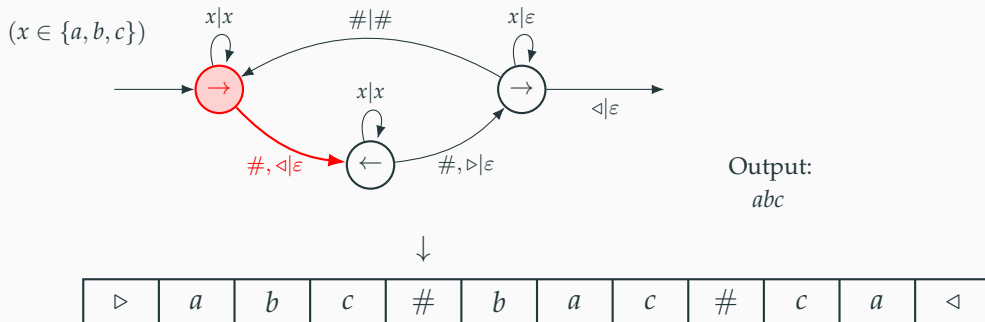
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



# Deterministic two-way transducers (2DFT)

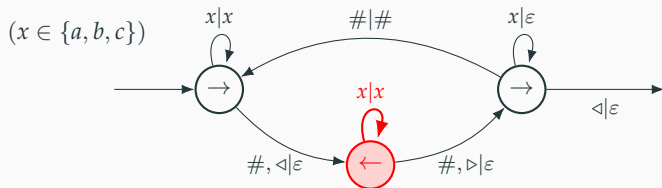
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

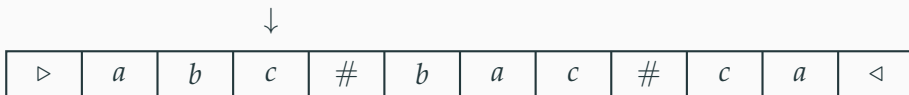
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abc*





# Deterministic two-way transducers (2DFT)

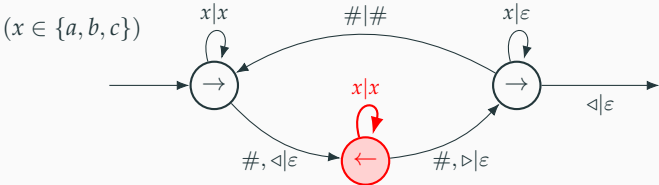
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

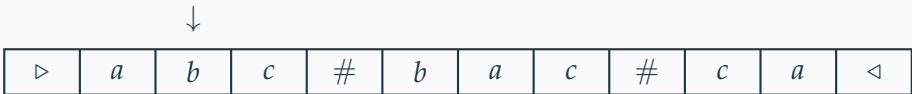
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
abcc



# Deterministic two-way transducers (2DFT)

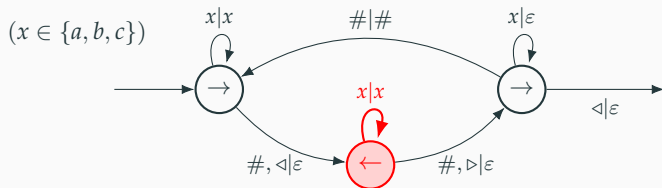
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

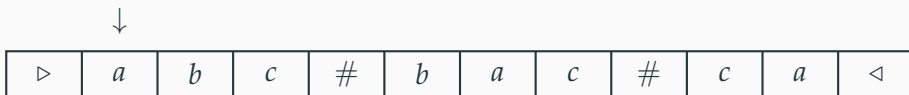
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccb*



# Deterministic two-way transducers (2DFT)

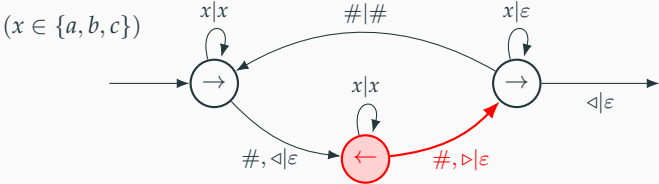
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

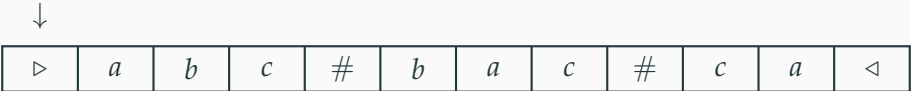
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba*



# Deterministic two-way transducers (2DFT)

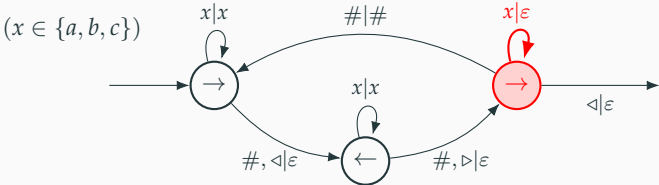
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

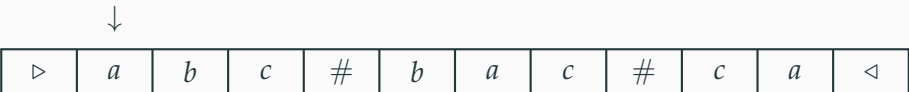
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba*



# Deterministic two-way transducers (2DFT)

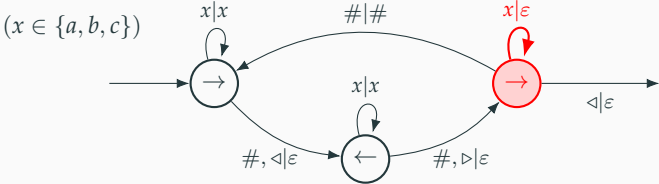
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

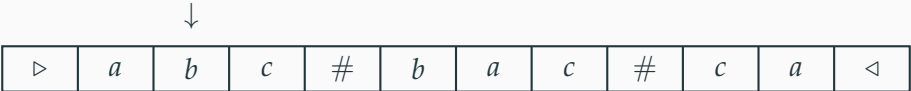
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba*



# Deterministic two-way transducers (2DFT)

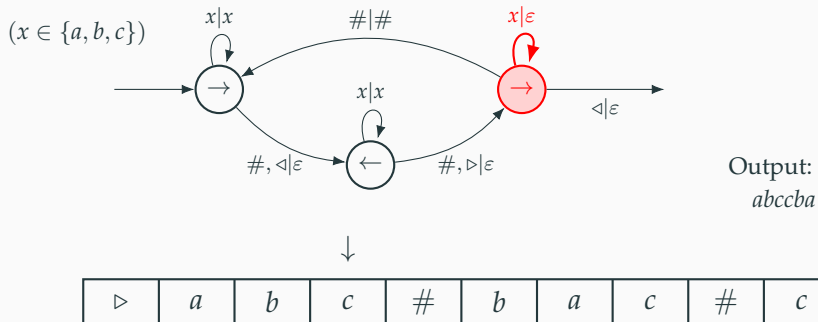
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



# Deterministic two-way transducers (2DFT)

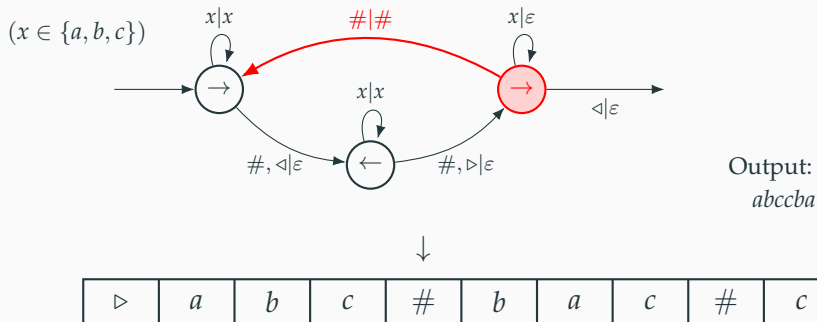
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



# Deterministic two-way transducers (2DFT)

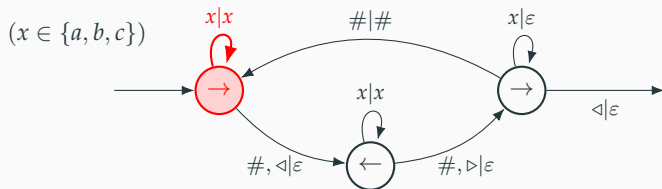
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

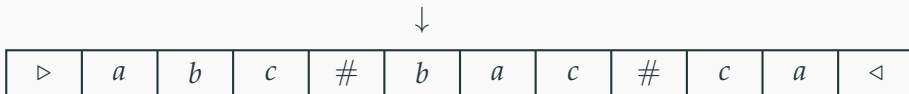
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#*





# Deterministic two-way transducers (2DFT)

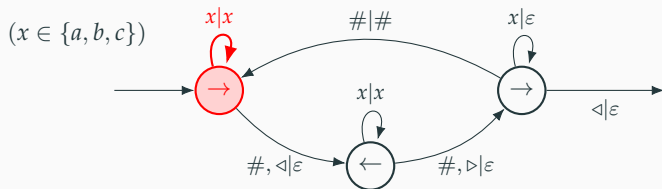
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

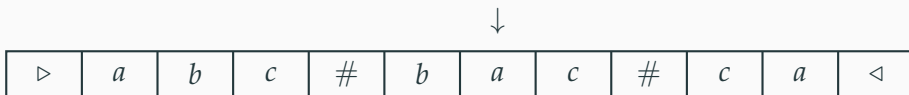
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#b*



# Deterministic two-way transducers (2DFT)

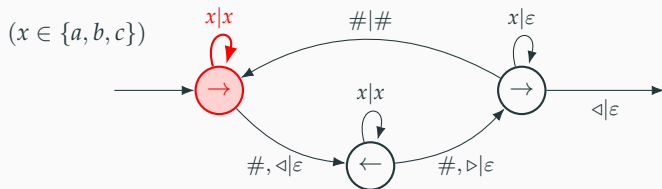
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

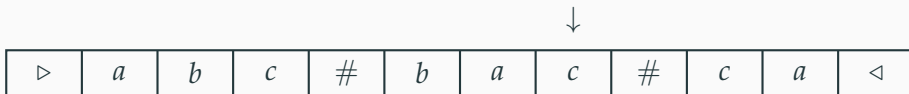
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#ba*



# Deterministic two-way transducers (2DFT)

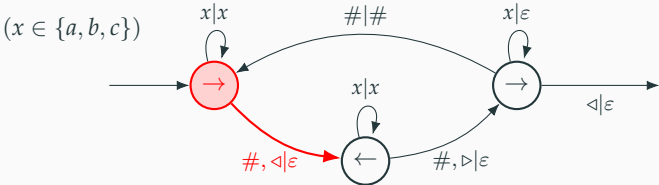
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
abccba#bac



▷	a	b	c	#	b	a	c	#	c	a	◁
---	---	---	---	---	---	---	---	---	---	---	---

# Deterministic two-way transducers (2DFT)

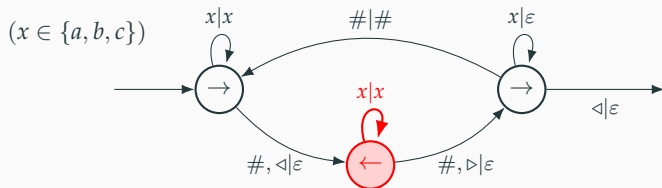
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

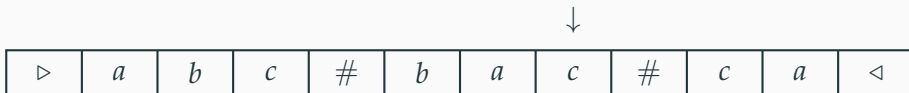
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bac*



# Deterministic two-way transducers (2DFT)

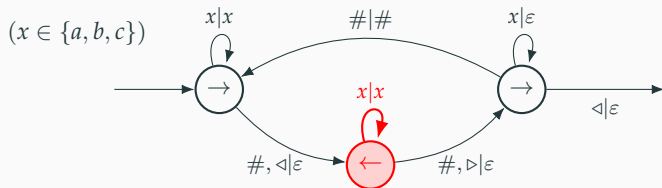
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

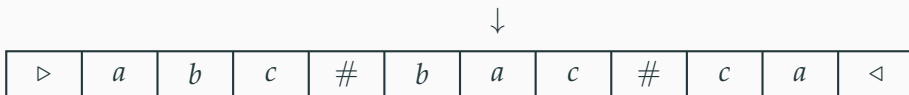
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bacc*



# Deterministic two-way transducers (2DFT)

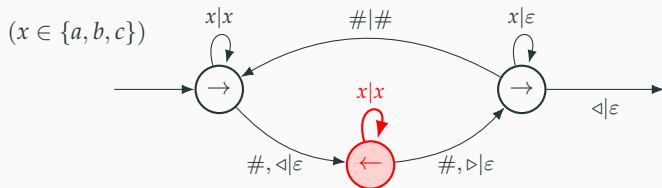
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

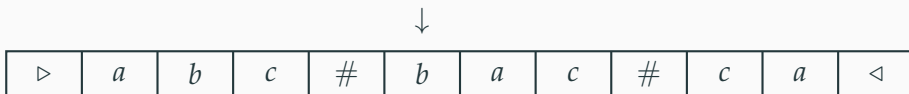
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bacca*



# Deterministic two-way transducers (2DFT)

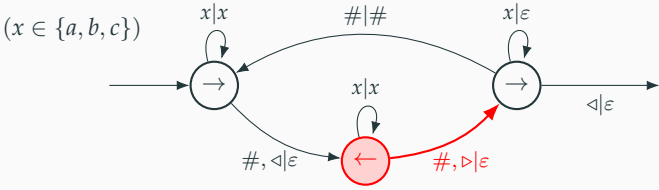
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

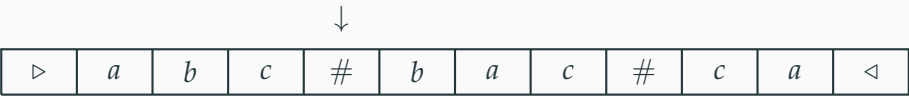
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bccab*



# Deterministic two-way transducers (2DFT)

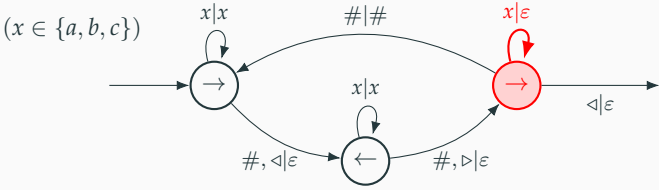
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

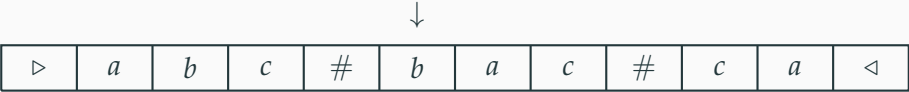
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bccab*





# Deterministic two-way transducers (2DFT)

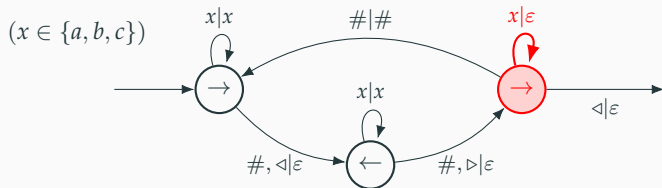
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

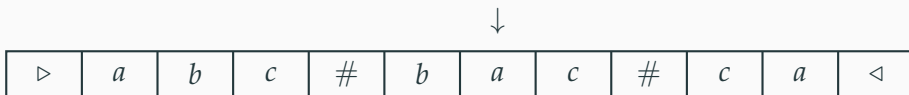
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bccab*



# Deterministic two-way transducers (2DFT)

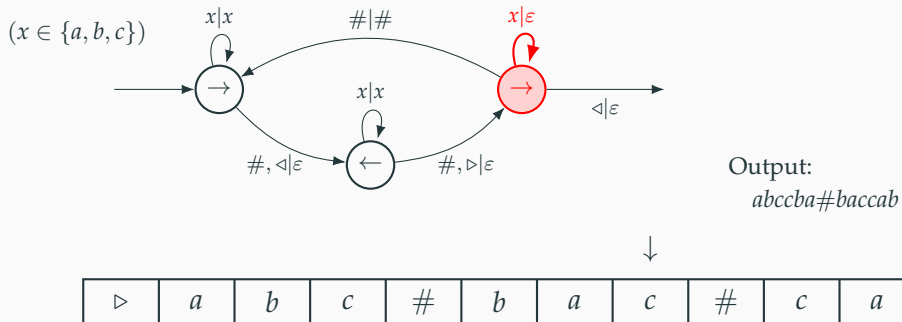
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



# Deterministic two-way transducers (2DFT)

Topic of this talk: certain basic computation models for string-to-string functions

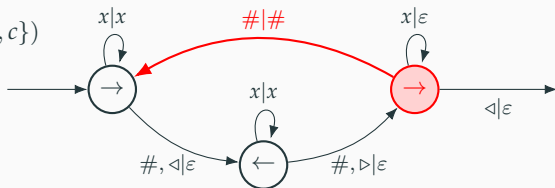
## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$

$(x \in \{a, b, c\})$



Output:

*abccba#bccab*

↓



# Deterministic two-way transducers (2DFT)

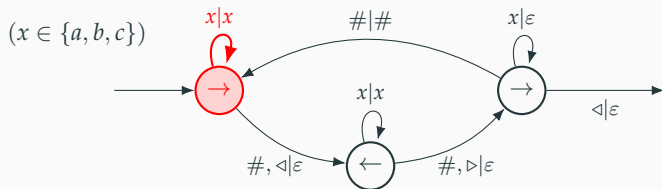
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bccab#*



# Deterministic two-way transducers (2DFT)

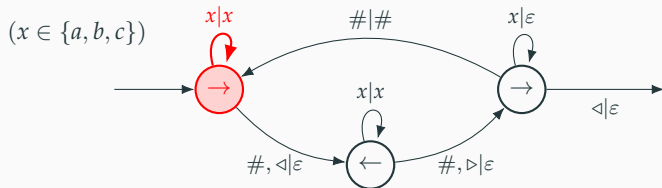
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bccab#c*



# Deterministic two-way transducers (2DFT)

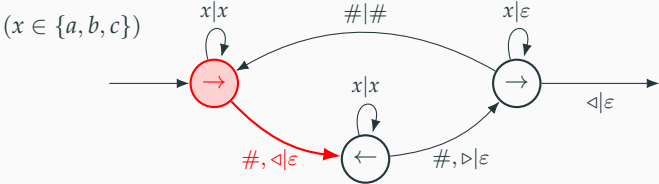
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

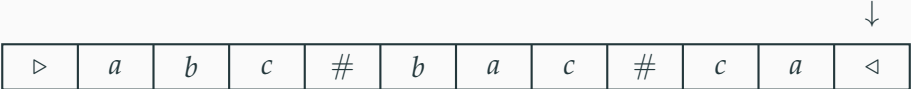
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bccab#ca*



# Deterministic two-way transducers (2DFT)

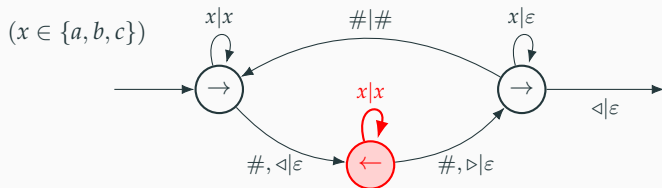
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bccab#ca*



# Deterministic two-way transducers (2DFT)

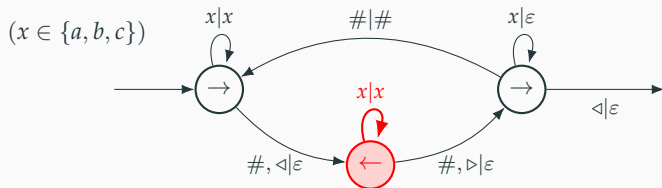
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bccab#caa*

↓





# Deterministic two-way transducers (2DFT)

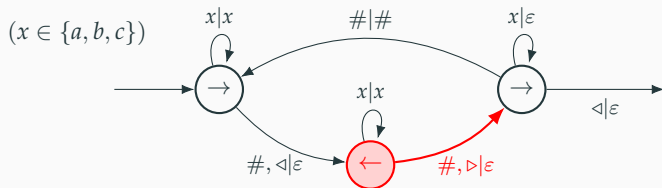
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
*abccba#bccab#caac*

↓



# Deterministic two-way transducers (2DFT)

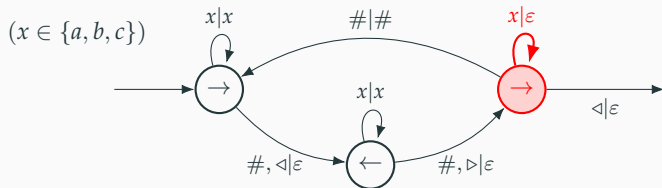
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:

*abccba#bccab#caac*

↓



# Deterministic two-way transducers (2DFT)

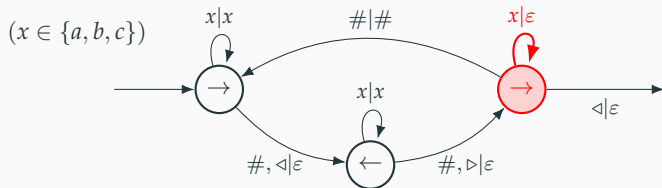
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1 \# \dots \# w_n &\longmapsto w_1 \cdot \text{reverse}(w_1) \# \dots \# w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:

*abccba#bccab#caac*

↓



# Deterministic two-way transducers (2DFT)

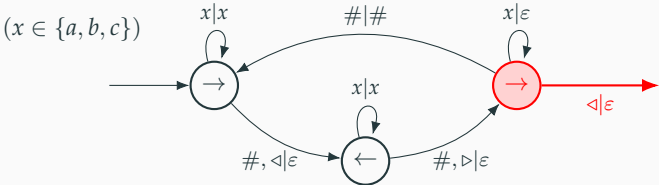
Topic of this talk: certain basic computation models for string-to-string functions

## Two-way transducers: executive summary

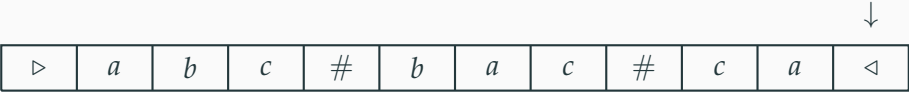
Finite set of states + bidirectional reading head + output produced from left to right

Example:

$$\begin{aligned} \text{mapPalin} : \{a, b, c, \#\}^* &\longrightarrow \{a, b, c, \#\}^* \\ w_1\# \dots \#w_n &\longmapsto w_1 \cdot \text{reverse}(w_1)\# \dots \#w_n \cdot \text{reverse}(w_n) \end{aligned}$$



Output:  
abccbba#baccab#caac



Functions  $\Sigma^* \rightarrow \Gamma^*$  definable by 2DFTs are called **regular functions**

## Properties of regular functions

- Linear growth:  $|f(w)| = O(|w|)$
- Closed under composition (if  $f: \Gamma^* \rightarrow \Sigma^*$  and  $g: \Sigma^* \rightarrow \Pi^*$  are regular then so is  $g \circ f$ )
- $L$  regular  $\implies f^{-1}(L)$  regular

## Alternative characterizations

- Via Monadic Second-Order logic (MSO transductions)
- Copyless streaming string transducers
- Various functional programming or regexp-like (declarative) formalisms
- (recent work of ours) Minimal linear  $\lambda$ -calculus and Church encodings [Nguyễn, Noûs, P. 2020]

Polyregular functions:

- A larger class of string-to-string transductions
- Garnered significant attention recently, starting with [Bojańczyk 2018]

## Properties

- *Polynomial* growth:  $|f(w)| = O(|w|^k)$
- $L$  regular  $\implies f^{-1}(L)$  regular
- Closed under composition

## Characterizations [Bojańczyk 2018; Bojańczyk, Kiefer & Lhote 2019]

- Multidimensional MSO interpretations
- Imperative nested loop programs
- Simply typed  $\lambda$ -calculus augmented with a list type and some list manipulation primitives
- Composition closure of [regular functions  $\cup$  “squaring with underlining”]

# Polyregular functions

Polyregular functions:

- A larger class of string-to-string transductions
- Garnered significant attention recently, starting with [Bojańczyk 2018]

## Properties

- *Polynomial* growth:  $|f(w)| = O(|w|^k)$
- $L$  regular  $\implies f^{-1}(L)$  regular
- Closed under composition

## Characterizations [Bojańczyk 2018; Bojańczyk, Kiefer & Lhote 2019]

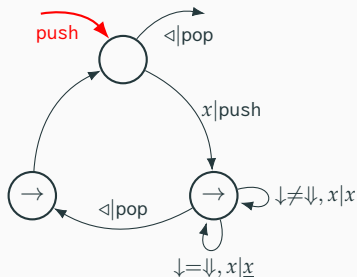
- Multidimensional MSO interpretations
- Imperative nested loop programs
- Simply typed  $\lambda$ -calculus augmented with a list type and some list manipulation primitives
- Composition closure of [regular functions  $\cup$  “squaring with underlining”]
- **$k$ -pebble string-to-string transducers**

## $k$ -pebble transducers: executive summary

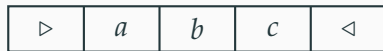
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} \underline{b} \underline{a} \underline{a} \underline{b} \end{aligned}$$



output =

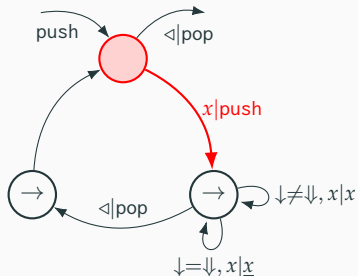


## $k$ -pebble transducers: executive summary

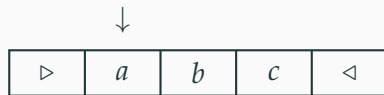
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



squaring :  $\Sigma^* \rightarrow (\Sigma \cup \underline{\Sigma})^*$   
 $aab \mapsto \underline{a} \underline{a} b \underline{a} \underline{a} b$



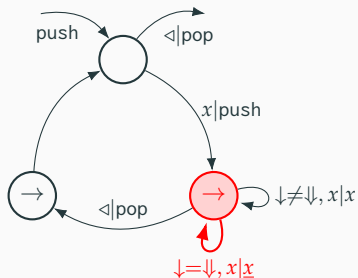
output =

## $k$ -pebble transducers: executive summary

Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

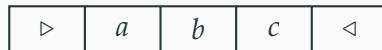
Example: “squaring with underlining” ( $k = 2$ )



squaring :  $\Sigma^* \rightarrow (\Sigma \cup \underline{\Sigma})^*$   
 $aab \mapsto \underline{a}a\underline{b}a\underline{a}b$

↓

↓



output =

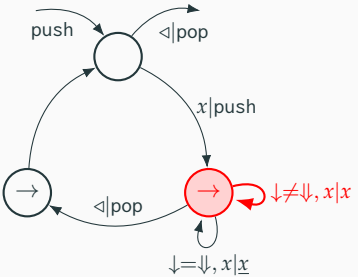
# Pebble transducers

## *k*-pebble transducers: executive summary

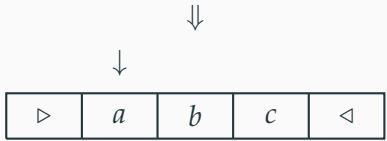
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} \underline{b} \underline{a} \underline{a} \underline{b} \end{aligned}$$



output = a

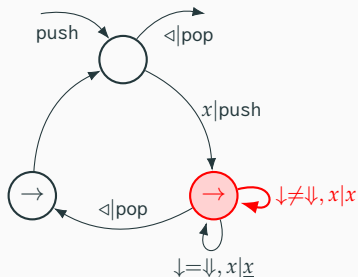
# Pebble transducers

## $k$ -pebble transducers: executive summary

Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

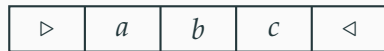
Example: “squaring with underlining” ( $k = 2$ )



squaring :  $\Sigma^* \rightarrow (\Sigma \cup \underline{\Sigma})^*$   
 $aab \mapsto \underline{a} \underline{a} b \underline{a} \underline{a} b$

$\Downarrow$

$\downarrow$



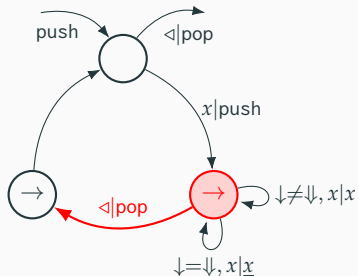
output =  $\underline{ab}$

## $k$ -pebble transducers: executive summary

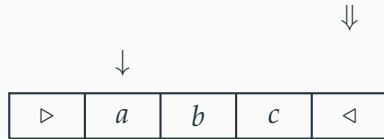
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



squaring :  $\Sigma^* \rightarrow (\Sigma \cup \underline{\Sigma})^*$   
 $aab \mapsto \underline{a} \underline{a} b \underline{a} \underline{a} b$



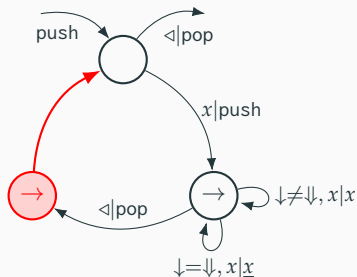
output = abc

## $k$ -pebble transducers: executive summary

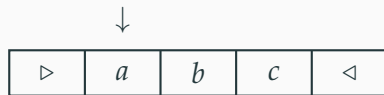
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} b \underline{a} \underline{a} b \end{aligned}$$



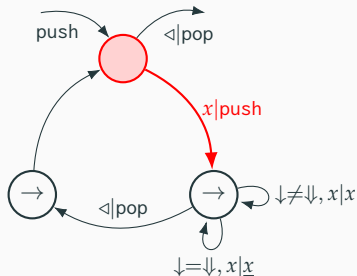
output = abc

## $k$ -pebble transducers: executive summary

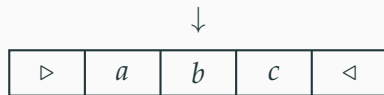
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



squaring :  $\Sigma^* \rightarrow (\Sigma \cup \underline{\Sigma})^*$   
 $aab \mapsto \underline{a} \underline{a} b \underline{a} \underline{a} b$



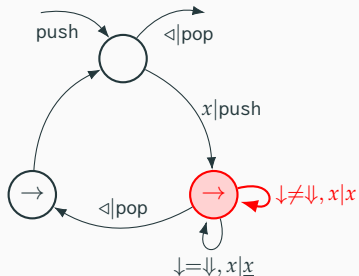
output =  $\underline{a} \underline{b} c$

## $k$ -pebble transducers: executive summary

Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )

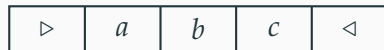


$$\text{squaring : } \Sigma^* \rightarrow (\Sigma \cup \underline{\Sigma})^*$$

$$aab \mapsto \underline{a} \underline{a} b \underline{a} \underline{a} b$$

$\Downarrow$

$\downarrow$



output =  $\underline{abc}$

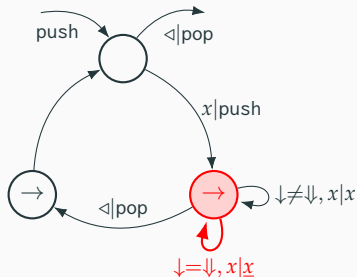


## $k$ -pebble transducers: executive summary

Finite set of states + a stack of two-way reading heads of height  $\leq k$

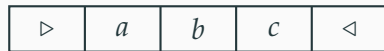
- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} \underline{b} \underline{a} \underline{a} \underline{b} \end{aligned}$$

↓  
↓



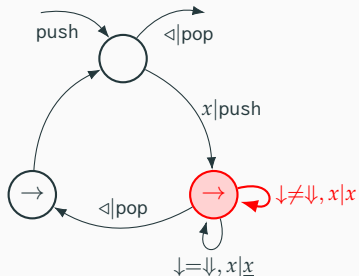
output = abca

## $k$ -pebble transducers: executive summary

Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

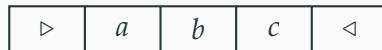
Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} \underline{b} \underline{a} \underline{a} \underline{b} \end{aligned}$$

$\Downarrow$

$\downarrow$



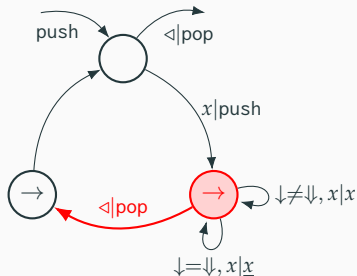
output =  $\underline{a} \underline{b} \underline{c} \underline{a} \underline{b}$

## $k$ -pebble transducers: executive summary

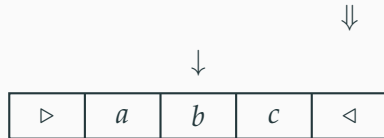
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} \underline{b} \underline{a} \underline{a} \underline{b} \end{aligned}$$



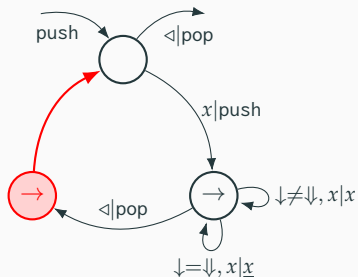
output = abcabc

## $k$ -pebble transducers: executive summary

Finite set of states + a stack of two-way reading heads of height  $\leq k$

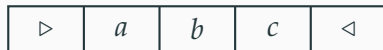
- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} b \underline{a} \underline{a} b \end{aligned}$$

↓



output =  $\underline{a} \underline{b} c \underline{a} \underline{b}$

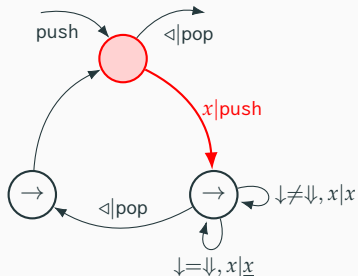
# Pebble transducers

## $k$ -pebble transducers: executive summary

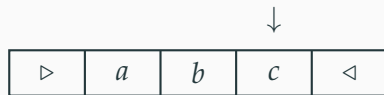
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} \underline{b} \underline{a} \underline{a} \underline{b} \end{aligned}$$



output = abcabc

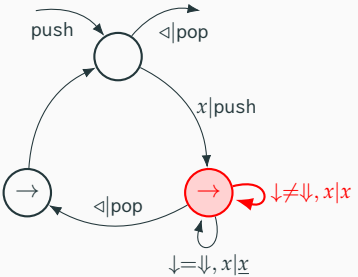
# Pebble transducers

## *k*-pebble transducers: executive summary

Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )

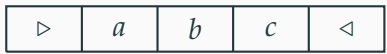


$$\text{squaring : } \Sigma^* \rightarrow (\Sigma \cup \underline{\Sigma})^*$$

$$aab \mapsto \underline{a} \underline{a} b \underline{a} \underline{a} b$$

$\Downarrow$

$\downarrow$



output = abcabc

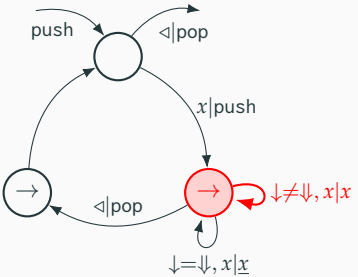
# Pebble transducers

## *k*-pebble transducers: executive summary

Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

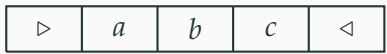
Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} \underline{b} \underline{a} \underline{a} \underline{b} \end{aligned}$$

↓

↓



output = abcab

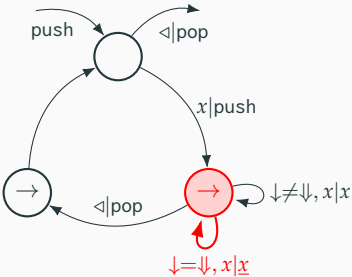
# Pebble transducers

## *k*-pebble transducers: executive summary

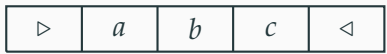
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} \underline{b} \underline{a} \underline{a} \underline{b} \end{aligned}$$



output = abcabc



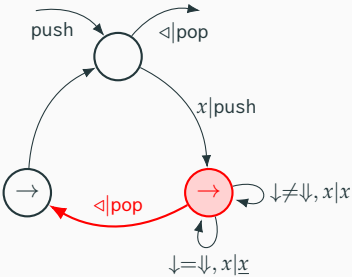
# Pebble transducers

## *k*-pebble transducers: executive summary

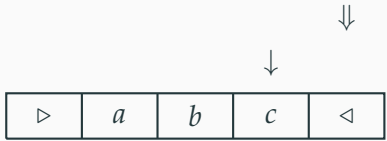
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} \underline{b} \underline{a} \underline{a} \underline{b} \end{aligned}$$



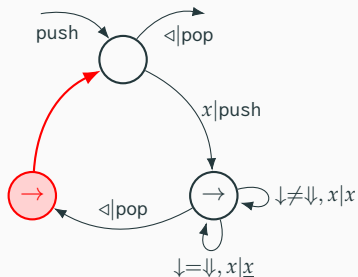
output = abcabc

## $k$ -pebble transducers: executive summary

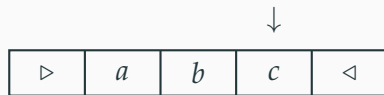
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} b \underline{a} \underline{a} b \end{aligned}$$



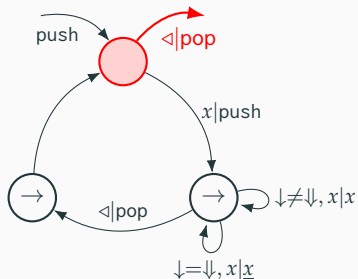
output = abcabc

## $k$ -pebble transducers: executive summary

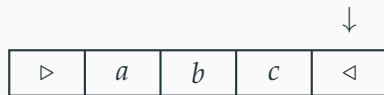
Finite set of states + a stack of two-way reading heads of height  $\leq k$

- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers  $\cong$  2DFTs

Example: “squaring with underlining” ( $k = 2$ )



$$\begin{aligned} \text{squaring : } \Sigma^* &\rightarrow (\Sigma \cup \underline{\Sigma})^* \\ aab &\mapsto \underline{a} \underline{a} \underline{b} \underline{a} \underline{a} \underline{b} \end{aligned}$$



output = abcabc

# Comparison-free pebble transducers

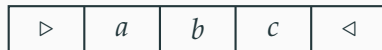
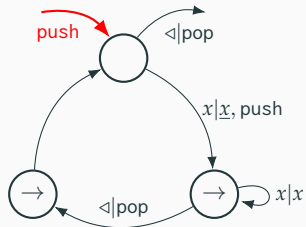
## Main question

What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”

$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}abb$$



output =

# Comparison-free pebble transducers

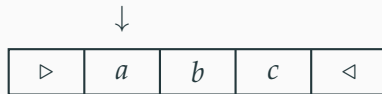
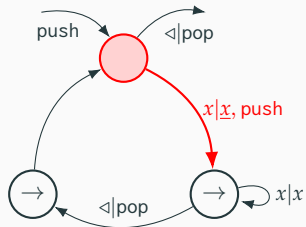
## Main question

What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”

$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}abb$$



output =

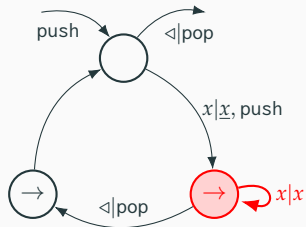
# Comparison-free pebble transducers

## Main question

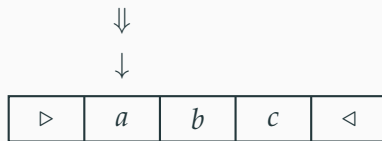
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}$$



output = a

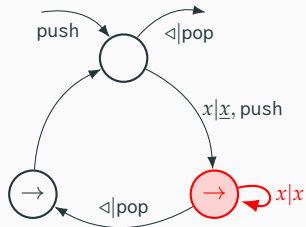
# Comparison-free pebble transducers

## Main question

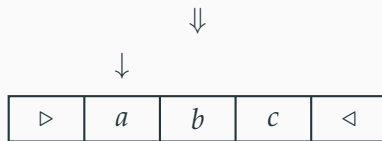
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}abb\underline{b}abb\underline{b}$$



output = aa

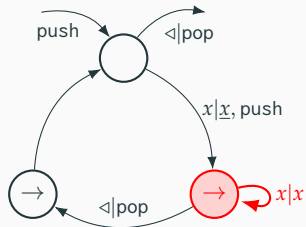
# Comparison-free pebble transducers

## Main question

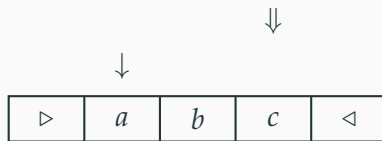
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}$$



output =  $\underline{a}ab$



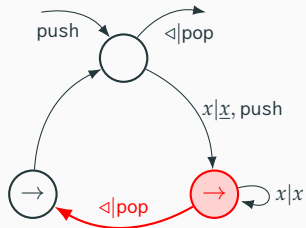
# Comparison-free pebble transducers

## Main question

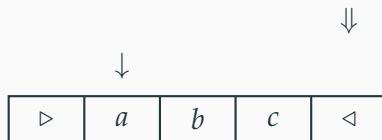
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}$



output =  $a$  $bc$

# Comparison-free pebble transducers

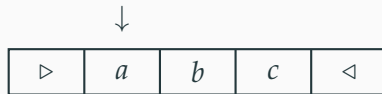
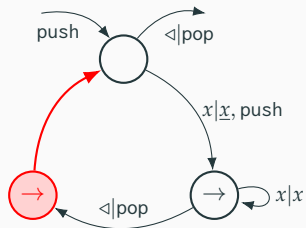
## Main question

What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”

$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}abb$$



output = abc

# Comparison-free pebble transducers

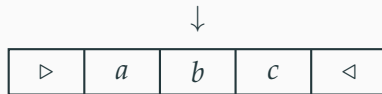
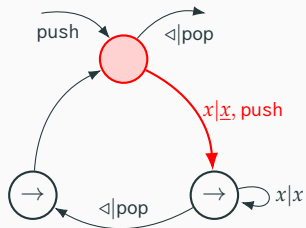
## Main question

What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”

$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}abb$$



output = abc

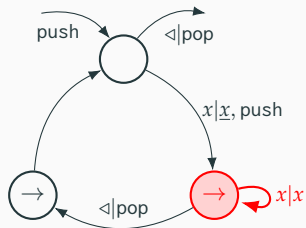
# Comparison-free pebble transducers

## Main question

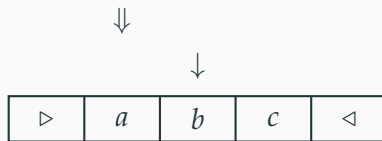
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}$$



output = abcb

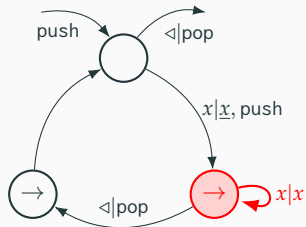
# Comparison-free pebble transducers

## Main question

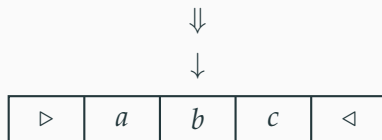
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}$$



$$\text{output} = \underline{a}b\underline{c}b\underline{a}$$

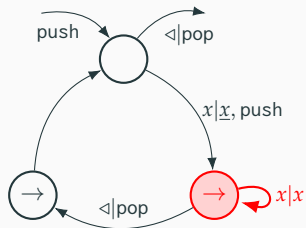
# Comparison-free pebble transducers

## Main question

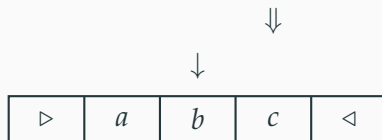
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}$$



output =  $\underline{a}b\underline{c}b\underline{a}b$

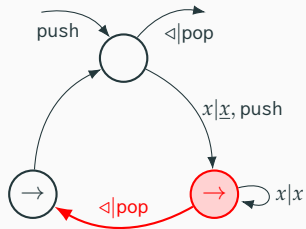
# Comparison-free pebble transducers

## Main question

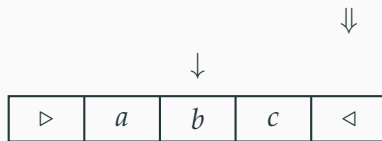
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}$$



output = abcbabc

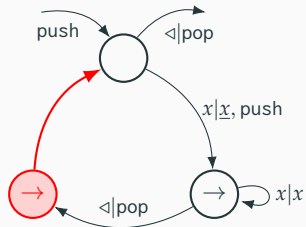
# Comparison-free pebble transducers

## Main question

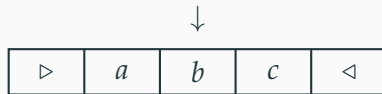
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}$$



output = abcbabc



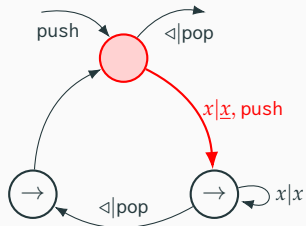
# Comparison-free pebble transducers

## Main question

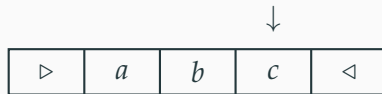
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}$$



output =  $\underline{a}b\underline{c}b\underline{a}b\underline{c}$

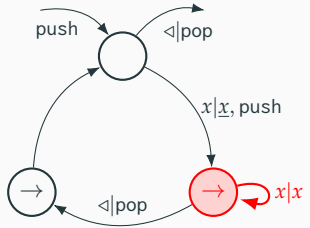
# Comparison-free pebble transducers

## Main question

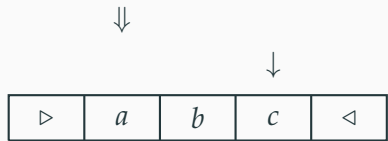
What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring"



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}abb$$



output = aabcbabcc

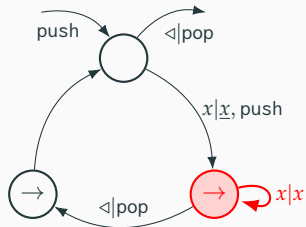
# Comparison-free pebble transducers

## Main question

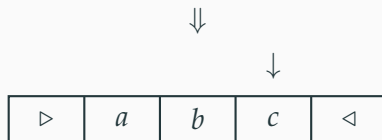
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}abb$$



output = abcbabcca

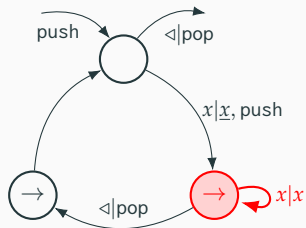
# Comparison-free pebble transducers

## Main question

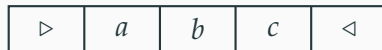
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}$$



$$\text{output} = \underline{a}b\underline{c}b\underline{a}b\underline{c}c\underline{a}b$$

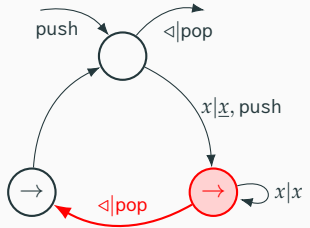
# Comparison-free pebble transducers

## Main question

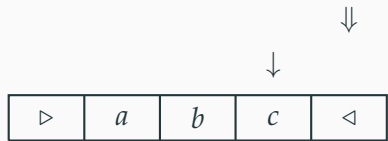
What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring"



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}abb$$



$$\text{output} = \underline{a}b\underline{c}b\underline{a}b\underline{c}c\underline{a}b\underline{c}$$

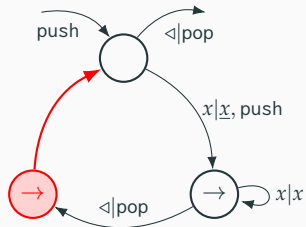
# Comparison-free pebble transducers

## Main question

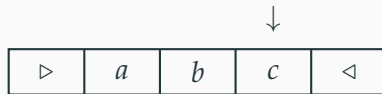
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}$$



output = abcbabccabc

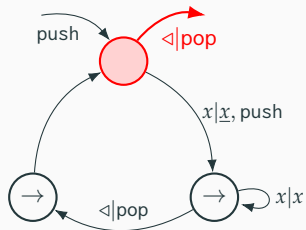
# Comparison-free pebble transducers

## Main question

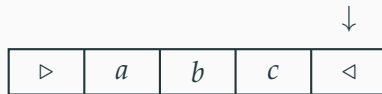
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”



$$\text{cfsquaring}(abb) = \underline{a}bb\underline{b}abb\underline{b}abb$$



output = abcbabccabc

# Comparison-free pebble transducers

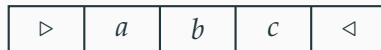
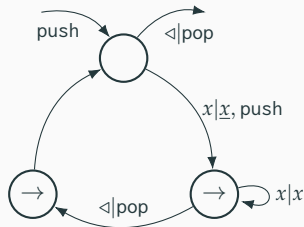
## Main question

What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”

$$\text{cfsquaring}(abb) = \underline{a}abb\underline{b}abb\underline{b}abb$$



output = aabcbabccabc

## Contributions

- Alternative characterizations
- Separation results
- Along the way: closure by composition, pebble minimization



## **Some alternative characterizations**

---

## Alternative characterization (1/2): composition by substitutions

### Definition (Composition by substitutions)

Let  $\Gamma, \Sigma$  and  $I$  be finite alphabets and  $f: \Gamma^* \rightarrow I^*$ ,  $g_i: \Gamma^* \rightarrow \Sigma^*$  and  $w \in \Gamma^*$ .

Define  $\text{CbS}(f, (g_i)_{i \in I})(w)$  so that, if  $f(w) = i_1 \dots i_k$ , then

$$\text{CbS}(f, (g_i)_{i \in I})(w) = g_{i_1}(w) \dots g_{i_k}(w)$$

E.g. for cfsquaring, we take  $f: \Sigma^* \rightarrow (\Sigma \cup \{X\})^*$ ,  $g_X, g_a: \Sigma^* \rightarrow (\Sigma \cup \underline{\Sigma})^*$  (for  $a \in \Sigma$ ) so that

$$f(abc) = aXbXcX \quad g_a(w) = \underline{a} \quad \text{and} \quad g_X(w) = w$$

- Note: both cfpolyreg and polyregular functions are closed under CbS

## Alternative characterization (1/2): composition by substitutions

### Definition (Composition by substitutions)

Let  $\Gamma, \Sigma$  and  $I$  be finite alphabets and  $f: \Gamma^* \rightarrow I^*$ ,  $g_i: \Gamma^* \rightarrow \Sigma^*$  and  $w \in \Gamma^*$ .

Define  $\text{CbS}(f, (g_i)_{i \in I})(w)$  so that, if  $f(w) = i_1 \dots i_k$ , then

$$\text{CbS}(f, (g_i)_{i \in I})(w) = g_{i_1}(w) \dots g_{i_k}(w)$$

E.g. for cfsquaring, we take  $f: \Sigma^* \rightarrow (\Sigma \cup \{X\})^*$ ,  $g_X, g_a: \Sigma^* \rightarrow (\Sigma \cup \underline{\Sigma})^*$  (for  $a \in \Sigma$ ) so that

$$f(abc) = aXbXcX \quad g_a(w) = \underline{a} \quad \text{and} \quad g_X(w) = w$$

- Note: both cfpolyreg and polyregular functions are closed under CbS

### Alternative definition of cfpolyregular functions

Smallest class such that

- Every regular function is cfpolyreg
- If  $f$  is regular and  $g_i$  is cfpolyreg for every  $i \in I$  then  $\text{CbS}(f, (g_i)_{i \in I})$  is cfpolyreg

## Alternative characterization (1/2): composition by substitutions

### Definition (Composition by substitutions)

Let  $\Gamma, \Sigma$  and  $I$  be finite alphabets and  $f: \Gamma^* \rightarrow I^*$ ,  $g_i: \Gamma^* \rightarrow \Sigma^*$  and  $w \in \Gamma^*$ .

Define  $\text{CbS}(f, (g_i)_{i \in I})(w)$  so that, if  $f(w) = i_1 \dots i_k$ , then

$$\text{CbS}(f, (g_i)_{i \in I})(w) = g_{i_1}(w) \dots g_{i_k}(w)$$

E.g. for cfsquaring, we take  $f: \Sigma^* \rightarrow (\Sigma \cup \{X\})^*$ ,  $g_X, g_a: \Sigma^* \rightarrow (\Sigma \cup \underline{\Sigma})^*$  (for  $a \in \Sigma$ ) so that

$$f(abc) = aXbXcX \quad g_a(w) = \underline{a} \quad \text{and} \quad g_X(w) = w$$

- Note: both cfpolyreg and polyregular functions are closed under CbS

### Alternative definition of cfpolyregular functions

Smallest class such that

- Every regular function is cfpolyreg
- If  $f$  is regular and  $g_i$  is cfpolyreg for every  $i \in I$  then  $\text{CbS}(f, (g_i)_{i \in I})$  is cfpolyreg
- More convenient to manipulate formally
- Tight link between the number of pebbles and the nesting of the CbS operator

We have an alternative characterization based on linear the  $\lambda$ -calculus

- Not presented in the paper, mostly based on [Nguyễn, Noûs, P. 2020]
- Hints at the following non-trivial theorem

(reproven with automata-theoretic tools in the paper with no references to the  $\lambda$ -calculus)

### Closure under composition

If  $f: \Sigma^* \rightarrow \Gamma^*$  and  $g: \Gamma^* \rightarrow \Delta^*$  are both cfpolyregular, so is  $g \circ f$ .

We have an alternative characterization based on linear the  $\lambda$ -calculus

- Not presented in the paper, mostly based on [Nguyễn, Noûs, P. 2020]
- Hints at the following non-trivial theorem

(reproven with automata-theoretic tools in the paper with no references to the  $\lambda$ -calculus)

### Closure under composition

If  $f: \Sigma^* \rightarrow \Gamma^*$  and  $g: \Gamma^* \rightarrow \Delta^*$  are both cfpolyregular, so is  $g \circ f$ .

Leads to a combinator-based definition.

### Alternative definition of cfpolyregular functions

Least class containing the regular functions, cfsquaring and closed under composition.

- Analogous to the case of general polyregular functions  
cfsquaring replaced by “squaring with underlining” in the above  $\rightarrow$  all polyregular functions
- Regular functions can also themselves be decomposed

**Not all polyregular functions are  
comparison-free**

---

### Theorem

The function  $f: a^n \in \{a\}^* \mapsto a\#aa\#\dots\#a^n$  is polyregular but not comparison-free.

Corollary: “squaring with underlining” is not CF.

### Theorem

$g: a^{n_1}\#\dots\#a^{n_k} \in \{a, \#\}^* \mapsto a^{n_1 \times n_1}\#\dots\#a^{n_k \times n_k}$  is polyregular but not comparison-free.

([Douéneau-Tabot 2021] proves a stronger result)



### Theorem

The function  $f: a^n \in \{a\}^* \mapsto a\#aa\#\dots\#a^n$  is polyregular but not comparison-free.

Corollary: “squaring with underlining” is not CF.

$f$  is also an *HDT0L transduction* ( $\iff$  computable by a copyful streaming string transducer / marble transducer [Douéneau-Tabot et al. 2020]). Therefore  $\text{HDT0L} \not\subseteq \text{cfpolyreg}$ ; conversely:

### Theorem

$w \in \Gamma^* \mapsto w^{|w|}$  is comparison-free polyregular, but when  $|\Gamma| \geq 2$ , it is not HDT0L.

Note: polynomially growing  $\text{HDT0L} \subset \text{polyreg}$

### Theorem

$g: a^{n_1}\#\dots\#a^{n_k} \in \{a, \#\}^* \mapsto a^{n_1 \times n_1}\#\dots\#a^{n_k \times n_k}$  is polyregular but not comparison-free.

([Douéneau-Tabot 2021] proves a stronger result)

### Theorem

The function  $f: a^n \in \{a\}^* \mapsto a\#aa\#\dots\#a^n$  is polyregular but not comparison-free.

Corollary: “squaring with underlining” is not CF.

$f$  is also an *HDT0L transduction* ( $\iff$  computable by a copyful streaming string transducer / marble transducer [Douéneau-Tabot et al. 2020]). Therefore HDT0L  $\not\subset$  cfpolyreg; conversely:

### Theorem

$w \in \Gamma^* \mapsto w^{|w|}$  is comparison-free polyregular, but when  $|\Gamma| \geq 2$ , it is not HDT0L.

Note: polynomially growing HDT0L  $\subset$  polyreg

### Theorem

$g: a^{n_1}\#\dots\#a^{n_k} \in \{a, \#\}^* \mapsto a^{n_1 \times n_1}\#\dots\#a^{n_k \times n_k}$  is polyregular but not comparison-free.

([Douéneau-Tabot 2021] proves a stronger result)

### Definition

For  $h: \Gamma^* \rightarrow \Sigma^*$ ,  $w_1, \dots, w_n \in \Gamma^*$  with  $\# \notin \Gamma$ ,  $\mathbf{map}(h)(w_1\#\dots\#w_n) = f(w_1)\#\dots\#f(w_n)$ .

$g = \mathbf{map}(w \mapsto w^{|w|})$  therefore comparison-free polyregular functions are *not* closed under  $\mathbf{map}$ , unlike regular and polyreg functions  
 $\rightarrow$  obstruction to characterizing cfpolyreg fn by list-processing functional programs (à la [Bojańczyk, Daviaud & Krishna 2018])

### Theorem

$g(a^{n_1} \# \dots \# a^{n_k}) = a^{n_1 \times n_1} \# \dots \# a^{n_k \times n_k}$  is not comparison-free polyregular.

Proof by contradiction: assume  $g$  is cfpolyreg.

First,  $|g(w)| = O(|w|^2)$  *therefore*  $g$  is computed by some 2-cf-pebble transducer.

## Separation proof idea for “map unary square” via pebble minimization

### Pebble minimization – major result of our paper

If  $f$  is cfpolyreg and  $|f(w)| = O(|w|^k)$  then some comparison-free  $k$ -pebble transducer computes  $f$ .

Very technical proof adapted from the analogous result for pebble transducers [Lhote 2020].

### Theorem

$g(a^{n_1} \# \dots \# a^{n_k}) = a^{n_1 \times n_1} \# \dots \# a^{n_k \times n_k}$  is not comparison-free polyregular.

Proof by contradiction: assume  $g$  is cfpolyreg.

First,  $|g(w)| = O(|w|^2)$  *therefore*  $g$  is computed by some 2-cf-pebble transducer.

## Separation proof idea for “map unary square” via pebble minimization

### Pebble minimization – major result of our paper

If  $f$  is cfpolyreg and  $|f(w)| = O(|w|^k)$  then some comparison-free  $k$ -pebble transducer computes  $f$ .

Very technical proof adapted from the analogous result for pebble transducers [Lhote 2020].

### Theorem

$g(a^{n_1} \# \dots \# a^{n_k}) = a^{n_1 \times n_1} \# \dots \# a^{n_k \times n_k}$  is not comparison-free polyregular.

Proof by contradiction: assume  $g$  is cfpolyreg.

First,  $|g(w)| = O(|w|^2)$  *therefore*  $g$  is computed by some 2-cf-pebble transducer. Equivalently, for some *finite*  $I$  and *regular* functions  $f$  and  $h_i$ ,

$$g = \text{CbS}(f, (h_i)_{i \in I}) \quad \text{i.e.} \quad f(w) = i_1 \dots i_m \implies g(w) = h_{i_1}(w) \dots h_{i_m}(w)$$

To conclude:

pumping argument + pigeonhole principle, exploiting the linear asymptotic growth

## Separation proof idea for “map unary square” via pebble minimization

### Pebble minimization – major result of our paper

If  $f$  is cfpolyreg and  $|f(w)| = O(|w|^k)$  then some comparison-free  $k$ -pebble transducer computes  $f$ .

Very technical proof adapted from the analogous result for pebble transducers [Lhote 2020].

### Theorem

$g(a^{n_1} \# \dots \# a^{n_k}) = a^{n_1 \times n_1} \# \dots \# a^{n_k \times n_k}$  is not comparison-free polyregular.

Proof by contradiction: assume  $g$  is cfpolyreg.

First,  $|g(w)| = O(|w|^2)$  *therefore*  $g$  is computed by some 2-cf-pebble transducer. Equivalently, for some *finite*  $I$  and *regular* functions  $f$  and  $h_i$ ,

$$g = \text{CbS}(f, (h_i)_{i \in I}) \quad \text{i.e.} \quad f(w) = i_1 \dots i_m \implies g(w) = h_{i_1}(w) \dots h_{i_m}(w)$$

To conclude:

pumping argument + pigeonhole principle, exploiting the linear asymptotic growth

Might be doable without pebble minimization, but convenient and of independent interest

## Separation proof idea continued: unary inputs

### Theorem

$f(a^n) = a\#aa\#\dots\#a^n$  is not cfpolyreg.

Observation:  $f(a^n)$  has the  $n$  maximal  $a$ -factors

$a \quad aa \quad \dots \quad a^n$

### Lemma

For any cfpolyreg  $g : \{a\}^* \rightarrow \Sigma^*$ , there are  $O(1)$  possible lengths for maximal  $a$ -factors in  $g(a^n)$ .

## Separation proof idea continued: unary inputs

### Theorem

$f(a^n) = a\#aa\#\dots\#a^n$  is not *cfpolyreg*.

Observation:  $f(a^n)$  has the  $n$  maximal  $a$ -factors

$a \quad aa \quad \dots \quad a^n$

### Lemma

For any *cfpolyreg*  $g : \{a\}^* \rightarrow \Sigma^*$ , there are  $O(1)$  possible lengths for maximal  $a$ -factors in  $g(a^n)$ .

In fact,  $\exists \mathcal{S} \subseteq \mathbb{Q}[X]$  *finite* such that  $\{P(n) \mid P \in \mathcal{S}\}$  contains  $\{\text{lengths of maximal } a\text{-factors of } g(a^n)\}$ , by structural induction on *poly-pumping sequences*.



## Separation proof idea continued: unary inputs

### Theorem

$f(a^n) = a\#aa\#\dots\#a^n$  is not *cfpolyreg*.

Observation:  $f(a^n)$  has the  $n$  maximal  $a$ -factors

$a \quad aa \quad \dots \quad a^n$

### Lemma

For any *cfpolyreg*  $g : \{a\}^* \rightarrow \Sigma^*$ , there are  $O(1)$  possible lengths for maximal  $a$ -factors in  $g(a^n)$ .

In fact,  $\exists \mathcal{S} \subseteq \mathbb{Q}[X]$  *finite* such that  $\{P(n) \mid P \in \mathcal{S}\}$  contains  $\{\text{lengths of maximal } a\text{-factors of } g(a^n)\}$ , by structural induction on *poly-pumping sequences*.

### Definition (poly-pumping sequence of words)

Smallest subclass of  $(\Sigma^*)^{\mathbb{N}}$

- Containing the constant sequences  $\alpha_n = w$
- Closed under concatenation  $\alpha_n = \beta_n \cdot \gamma_n$
- Closed under "iteration"  $\alpha_n = (\beta_n)^n$

## Separation proof idea continued: unary inputs

### Theorem

$f(a^n) = a\#aa\#\dots\#a^n$  is not *cfpolyreg*.

Observation:  $f(a^n)$  has the  $n$  maximal  $a$ -factors

$a \quad aa \quad \dots \quad a^n$

### Lemma

For any *cfpolyreg*  $g : \{a\}^* \rightarrow \Sigma^*$ , there are  $O(1)$  possible lengths for maximal  $a$ -factors in  $g(a^n)$ .

In fact,  $\exists \mathcal{S} \subseteq \mathbb{Q}[X]$  *finite* such that  $\{P(n) \mid P \in \mathcal{S}\}$  contains  $\{\text{lengths of maximal } a\text{-factors of } g(a^n)\}$ , by structural induction on *poly-pumping sequences*.

### Definition (poly-pumping sequence of words)

Smallest subclass of  $(\Sigma^*)^{\mathbb{N}}$

- Containing the constant sequences  $\alpha_n = w$
- Closed under concatenation  $\alpha_n = \beta_n \cdot \gamma_n$
- Closed under “iteration”  $\alpha_n = (\beta_n)^n$

### Theorem (cfpolyreg with unary input)

$f : \{a\}^* \rightarrow \Sigma^*$  is comparison-free polyregular if and only if  $\exists p \in \mathbb{N}$  such that  $(f(a^{(n+1)^{p+m}}))_{n \in \mathbb{N}}$  is poly-pumping for every  $m < p$ .

→ “ultimately periodic combinations” (u.p.c.)

## Separation proof idea continued: unary inputs

### Theorem

$f(a^n) = a\#aa\#\dots\#a^n$  is not cfpolyreg.

Observation:  $f(a^n)$  has the  $n$  maximal  $a$ -factors

$a \quad aa \quad \dots \quad a^n$

### Lemma

For any cfpolyreg  $g : \{a\}^* \rightarrow \Sigma^*$ , there are  $O(1)$  possible lengths for maximal  $a$ -factors in  $g(a^n)$ .

In fact,  $\exists \mathcal{S} \subseteq \mathbb{Q}[X]$  *finite* such that  $\{P(n) \mid P \in \mathcal{S}\}$  contains  $\{\text{lengths of maximal } a\text{-factors of } g(a^n)\}$ , by structural induction on *poly-pumping sequences*.

- Regular word sequences are u.p.c. of pumping sequences  $(u_0(v_1)^n \dots (v_l)^n u_l)_{n \in \mathbb{N}}$  [Choffrut 2017]  
Proof idea: find an idempotent in a suitable transition monoid of your favorite machine model for reg fn
- Proof for general cfpolyreg sequences: induction on the CbS-based definition

### Definition (poly-pumping sequence of words)

Smallest subclass of  $(\Sigma^*)^{\mathbb{N}}$

- Containing the constant sequences  $\alpha_n = w$
- Closed under concatenation  $\alpha_n = \beta_n \cdot \gamma_n$
- Closed under “iteration”  $\alpha_n = (\beta_n)^n$

### Theorem (cfpolyreg with unary input)

$f : \{a\}^* \rightarrow \Sigma^*$  is comparison-free polyregular if and only if  $\exists p \in \mathbb{N}$  such that  $(f(a^{(n+1)^{p+m}}))_{n \in \mathbb{N}}$  is poly-pumping for every  $m < p$ .

→ “ultimately periodic combinations” (u.p.c.)

## Further topics

---

### First-order (FO)-regular functions

Robust subclass of regular functions; several characterizations:

- Logic: replace MSO by first-order logic
- 2DFT with *aperiodic* monoid of behaviors
- Functional programming or regexp-like e.g. [Dartois, Gastin & Krishna 2021]

↪ Analogous class of FO-polyregular.

### First-order (FO)-regular functions

Robust subclass of regular functions; several characterizations:

- Logic: replace MSO by first-order logic
- 2DFT with *aperiodic* monoid of behaviors
- Functional programming or regexp-like e.g. [Dartois, Gastin & Krishna 2021]

↪ Analogous class of FO-polyregular.      What about cfpolyregular functions?

### First-order (FO)-regular functions

Robust subclass of regular functions; several characterizations:

- Logic: replace MSO by first-order logic
- 2DFT with *aperiodic* monoid of behaviors
- Functional programming or regexp-like e.g. [Dartois, Gastin & Krishna 2021]

↪ Analogous class of FO-polyregular.      What about cfpolyregular functions?

### Definition (First-order comparison-free polyregular functions)

FO-cfpolyreg = smallest class such that

- Every FO-regular function is FO-cfpolyreg
- If  $f$  is FO-regular and  $g_i$  is FO-cfpolyreg ( $\forall i \in I$ ), then  $\text{CbS}(f, (g_i)_{i \in I})$  is FO-cfpolyreg

### First-order (FO)-regular functions

Robust subclass of regular functions; several characterizations:

- Logic: replace MSO by first-order logic
- 2DFT with *aperiodic* monoid of behaviors
- Functional programming or regexp-like e.g. [Dartois, Gastin & Krishna 2021]

↪ Analogous class of FO-polyregular.      What about cfpolyregular functions?

### Definition (First-order comparison-free polyregular functions)

FO-cfpolyreg = smallest class such that

- Every FO-regular function is FO-cfpolyreg
- If  $f$  is FO-regular and  $g_i$  is FO-cfpolyreg ( $\forall i \in I$ ), then  $\text{CbS}(f, (g_i)_{i \in I})$  is FO-cfpolyreg

Other characterizations?



## Logical characterization of FO-cfpolyregular functions

Let  $\mathfrak{M} : \{\text{words}\} \rightarrow \{\text{finite models}\}$  be as usual.

For  $\mathfrak{U} = (U, R, \dots)$ , let  $\mathfrak{U}^k = (U^k, R_1, \dots, R_k, \dots)$  where  $R_i(x_1, \dots, x_m) :\Leftrightarrow R(\pi_i(x_1), \dots, \pi_i(x_m))$ .

### Conjecture

$f$  is *first-order* comparison-free polyregular if and only there exist  $k \in \mathbb{N}$  and a FO transduction  $\varphi$  s.t.

$$\forall w. \mathfrak{M}(f(w)) \simeq \varphi\left(\mathfrak{M}(w)^k\right)$$

## Logical characterization of FO-cfpolyregular functions

Let  $\mathfrak{M} : \{\text{words}\} \rightarrow \{\text{finite models}\}$  be as usual.

For  $\mathfrak{U} = (U, R, \dots)$ , let  $\mathfrak{U}^k = (U^k, R_1, \dots, R_k, \dots)$  where  $R_i(x_1, \dots, x_m) :\Leftrightarrow R(\pi_i(x_1), \dots, \pi_i(x_m))$ .

### Conjecture

$f$  is *first-order* comparison-free polyregular if and only there exist  $k \in \mathbb{N}$  and a FO transduction  $\varphi$  s.t.

$$\forall w. \mathfrak{M}(f(w)) \simeq \varphi\left(\mathfrak{M}(w)^k\right)$$

- Arguably leads to a logical characterization of general cfpolyregular as

FO-cfpolyregular functions     $\circ$     regular functions    =    cfpolyregular functions

## Logical characterization of FO-cfpolyregular functions

Let  $\mathfrak{M} : \{\text{words}\} \rightarrow \{\text{finite models}\}$  be as usual.

For  $\mathfrak{U} = (U, R, \dots)$ , let  $\mathfrak{U}^k = (U^k, R_1, \dots, R_k, \dots)$  where  $R_i(x_1, \dots, x_m) :\Leftrightarrow R(\pi_i(x_1), \dots, \pi_i(x_m))$ .

### Conjecture

$f$  is *first-order* comparison-free polyregular if and only there exist  $k \in \mathbb{N}$  and a FO transduction  $\varphi$  s.t.

$$\forall w. \mathfrak{M}(f(w)) \simeq \varphi\left(\mathfrak{M}(w)^k\right)$$

- Arguably leads to a logical characterization of general cfpolyregular as

$$\text{FO-cfpolyregular functions} \quad \circ \quad \text{regular functions} \quad = \quad \text{cfpolyregular functions}$$

- Equivalences with other candidates characterizing FO-cfpolyregular: apparently easier

E.g., FO-cfpolyregular = closure under  $\circ$  of FO-regular and cfsquaring

A few relevant directions:

- Extending this class to tree-to-tree functions and look for characterizations

A linear  $\lambda$ -calculus characterization matches an analogue of the CbS definition

A few relevant directions:

- Extending this class to tree-to-tree functions and look for characterizations

A linear  $\lambda$ -calculus characterization matches an analogue of the CbS definition

- Separation and membership problems, in the spirit of:

### Theorem [Douéneau-Tabot 2021]

There is an algorithm with

- **Input:** a pebble transducer implementing a function  $f$  with quadratic growth
- **Output:** a comparison-free transducer implementing  $f$ , or an error if there is none

A few relevant directions:

- Extending this class to tree-to-tree functions and look for characterizations

A linear  $\lambda$ -calculus characterization matches an analogue of the CbS definition

- Separation and membership problems, in the spirit of:

### Theorem [Douéneau-Tabot 2021]

There is an algorithm with

- **Input:** a pebble transducer implementing a function  $f$  with quadratic growth
- **Output:** a comparison-free transducer implementing  $f$ , or an error if there is none

- Non-commutative linear  $\lambda$ -calculus characterization for the FO case

## Conclusion

---

A new(?) class of string-to-string functions: *comparison-free polyregular functions*.

### Equivalent definitions

- By comparison-free pebble transducers
  - Inductively (composition by substitution)
  - **As the composition closure of regular functions** +  $\text{cfsquaring}(abc) = \underline{a}abc\underline{b}ab\underline{c}cabc$
- 
- $L$  regular language  $\implies f^{-1}(L)$  also regular
  - Polynomial growth:  $|f(w)| = O(|w|^k)$ 
    - **pebble minimization theorem:**  $k =$  number of heads necessary to compute  $f$
  - Strictly included in polyregular functions
    - $a^n \mapsto a\#aa\#\dots\#a^n$  and “map unary square” are polyregular but not cfpolyreg
    - for  $\{a\}^* \rightarrow \{a\}^*$  cfpolyreg = polyreg
  - Incomparable with polynomial HDT0L transductions
    - $a^n \mapsto a\#aa\#\dots\#a^n$  not cfpolyreg but HDT0L
    - $w \mapsto w^{|w|}$  is cfpolyreg but not HDT0L
  - Well-behaved first-order counterpart



A new(?) class of string-to-string functions: *comparison-free polyregular functions*.

### Equivalent definitions

- By comparison-free pebble transducers
  - Inductively (composition by substitution)
  - **As the composition closure of regular functions** +  $\text{cfsquaring}(abc) = \underline{a}abc\underline{b}ab\underline{c}cabc$
- 
- $L$  regular language  $\implies f^{-1}(L)$  also regular
  - Polynomial growth:  $|f(w)| = O(|w|^k)$ 
    - **pebble minimization theorem:**  $k =$  number of heads necessary to compute  $f$
  - Strictly included in polyregular functions
    - $a^n \mapsto a\#aa\#\dots\#a^n$  and “map unary square” are polyregular but not cfpolyreg
    - for  $\{a\}^* \rightarrow \{a\}^*$  cfpolyreg = polyreg
  - Incomparable with polynomial HDT0L transductions
    - $a^n \mapsto a\#aa\#\dots\#a^n$  not cfpolyreg but HDT0L
    - $w \mapsto w^{|w|}$  is cfpolyreg but not HDT0L
  - Well-behaved first-order counterpart

Thanks for watching! We'll be happy to take questions