Comparison-free polyregular functions

Nguyễn Lê Thành Dũng (a.k.a. Tito) --- nltd@nguyentito.eu
Laboratoire d'informatique de Paris Nord, Villetaneuse, France

Camille Noûs --- https://www.cogitamus.fr/camilleen.html

Cécilia Pradic --- pierre.pradic@cs.ox.ac.uk
University of Oxford, United Kingdom

ICALP 2021 --- track B

Introduction: string-to-string transducers

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

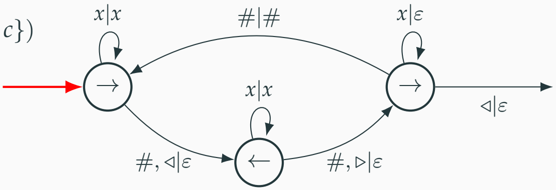
Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin} : \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:

$\triangleright$	a	b	c	$\#$	b	a	c	$\#$	c	a	$\triangleleft$

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

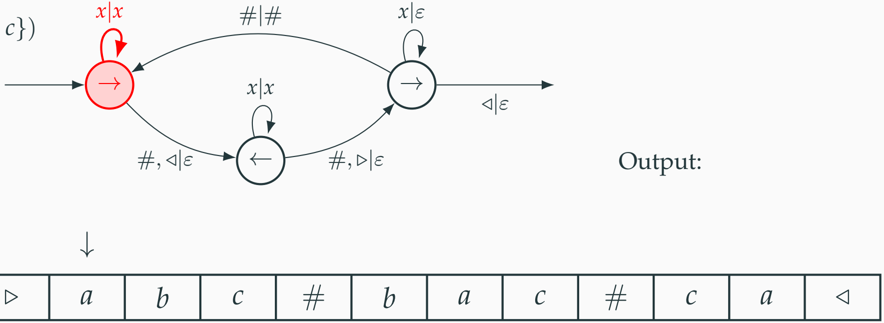
Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin}: \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$$

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



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

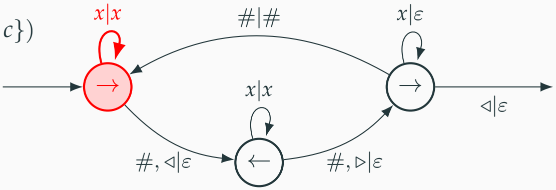
Two-way transducers: executive summary

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

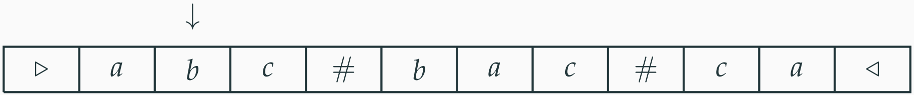
Example:

$$\begin{array}{llll} \texttt{mapPalin}: & \{a,b,c,\#\}^* & \longrightarrow & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \longmapsto & w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n) \end{array}$$

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



Output:
a



Deterministic two-way transducers (2DFT)

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

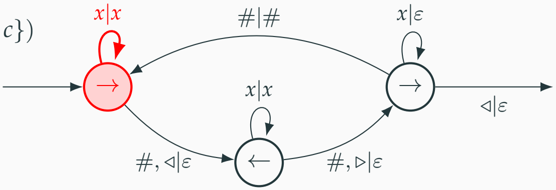
Two-way transducers: executive summary

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

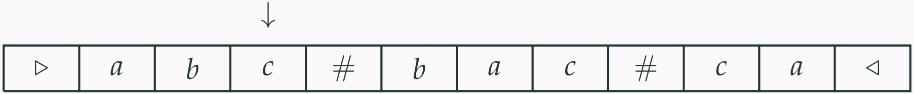
Example:

$$\texttt{mapPalin}: \quad \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:
ab



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

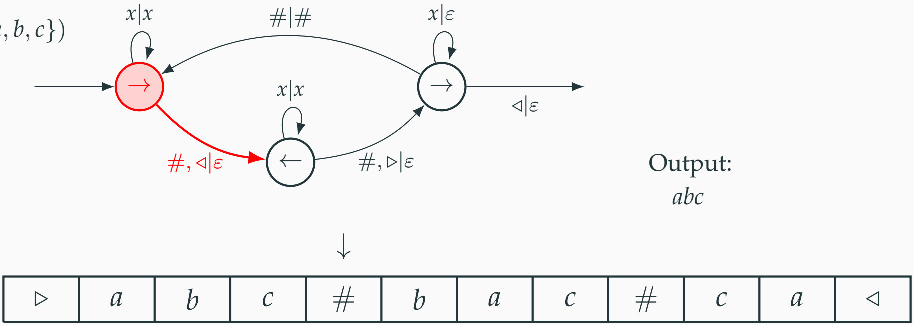
Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin} : \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



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

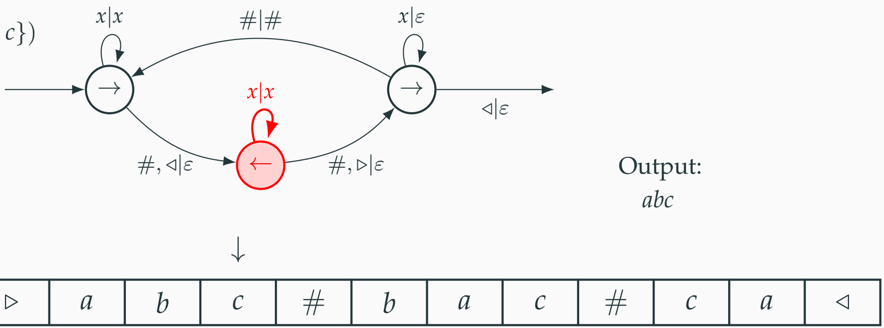
Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin}: \quad \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



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

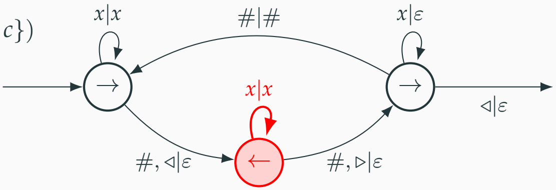
Two-way transducers: executive summary

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

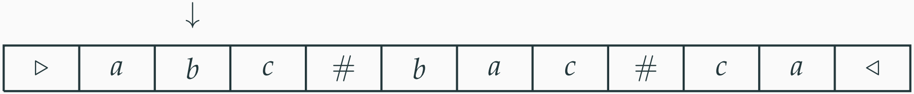
Example:

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

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



Output:
abcc



Deterministic two-way transducers (2DFT)

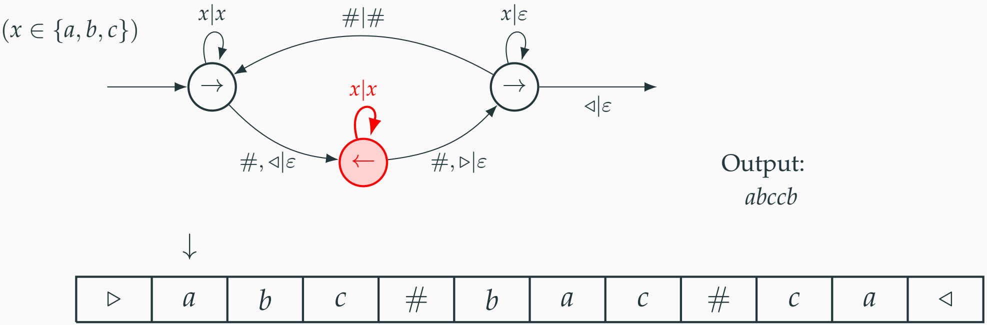
Topic of this talk: certain basic computation models for string-to-string functions

Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin}: \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$



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

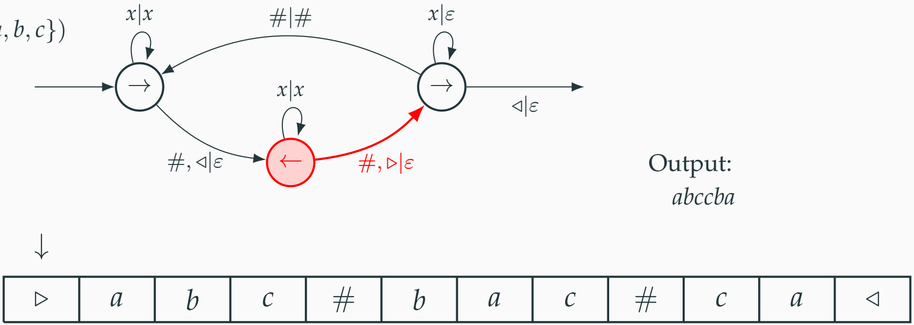
Two-way transducers: executive summary

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

Example:

$$\begin{array}{rccl} \texttt{mapPalin}: & \{a,b,c,\#\}^* & \longrightarrow & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \longmapsto & w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n) \end{array}$$

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



Deterministic two-way transducers (2DFT)

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

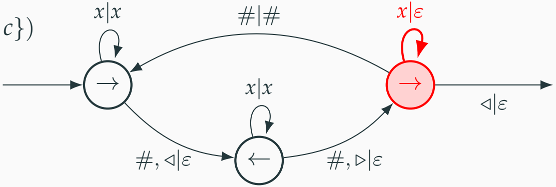
Two-way transducers: executive summary

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

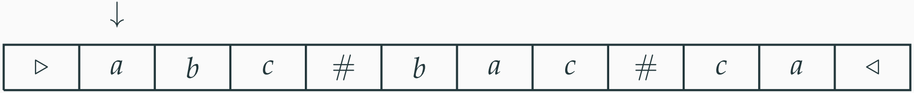
Example:

$$\texttt{mapPalin}: \quad \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:
abccba



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

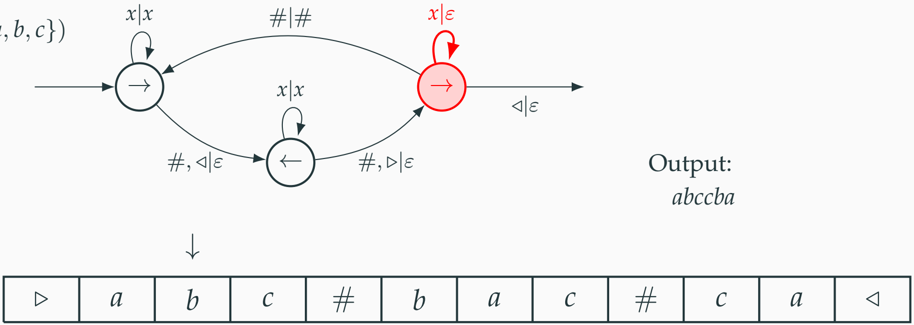
Two-way transducers: executive summary

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

Example:

$$\begin{array}{llcl} \texttt{mapPalin}: & \{a,b,c,\#\}^* & \longrightarrow & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \longmapsto & w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n) \end{array}$$

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



Deterministic two-way transducers (2DFT)

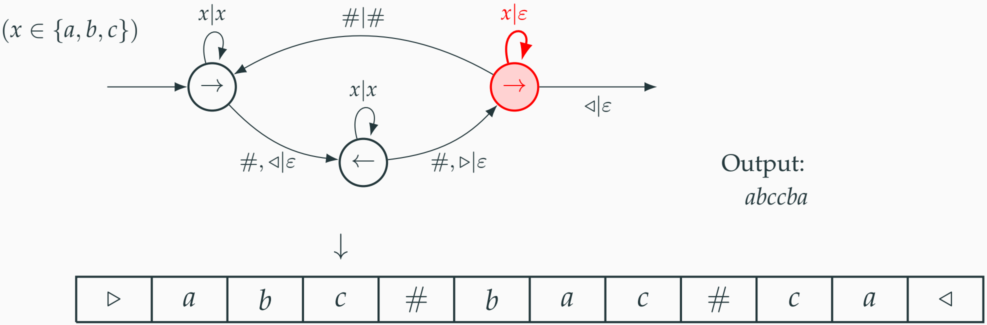
Topic of this talk: certain basic computation models for string-to-string functions

Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin}: \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$$



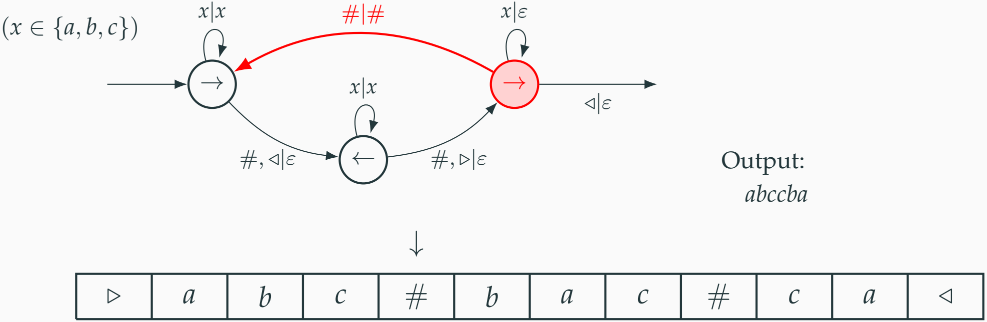
Topic of this talk: certain basic computation models for string-to-string functions

Two-way transducers: executive summary

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

Example:

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



Output:
abccba

Deterministic two-way transducers (2DFT)

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

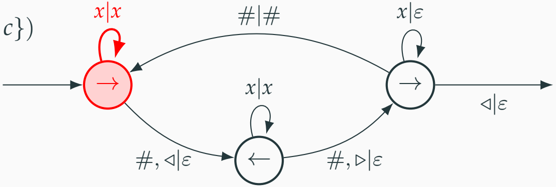
Two-way transducers: executive summary

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

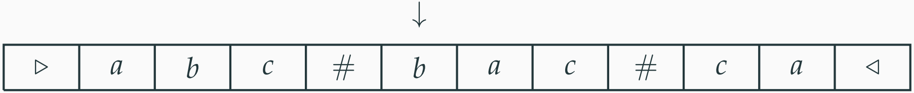
Example:

$$\texttt{mapPalin}: \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:
abccba#



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

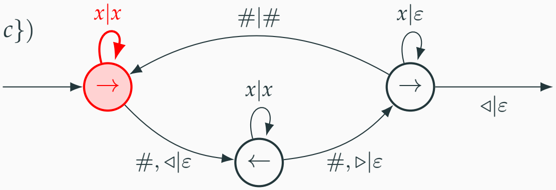
Two-way transducers: executive summary

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

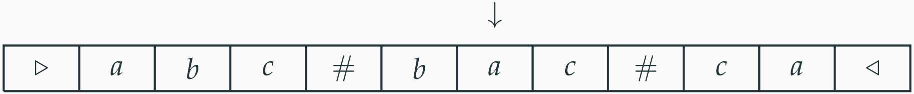
Example:

$$\texttt{mapPalin} : \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:
abccba#b



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

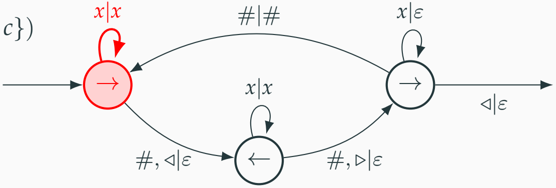
Two-way transducers: executive summary

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

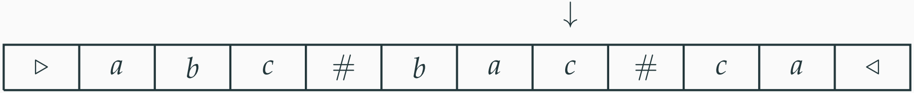
Example:

$$\texttt{mapPalin}: \quad \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:
$abccba\#ba$



Deterministic two-way transducers (2DFT)

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

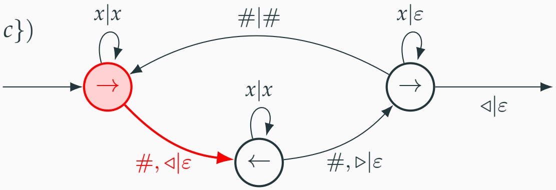
Two-way transducers: executive summary

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

Example:

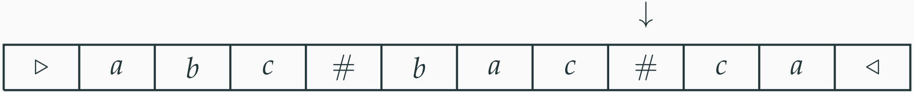
$$\texttt{mapPalin} : \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:

abccba#bac



Deterministic two-way transducers (2DFT)

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

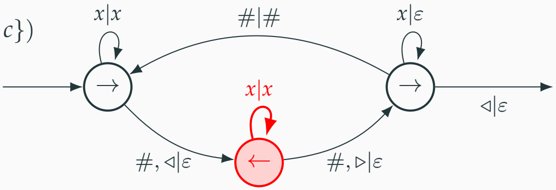
Two-way transducers: executive summary

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

Example:

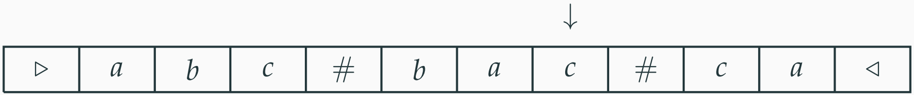
$$\texttt{mapPalin} : \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:

abccba#bac



Deterministic two-way transducers (2DFT)

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

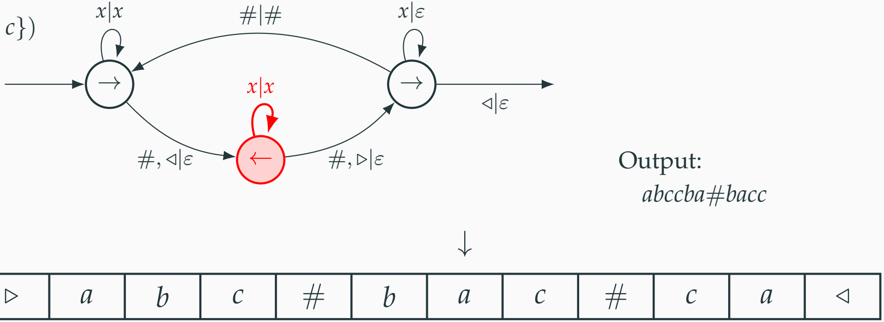
Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin}: \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1\cdot\texttt{reverse}(w_1)\#\ldots\#w_n\cdot\texttt{reverse}(w_n)$$

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



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

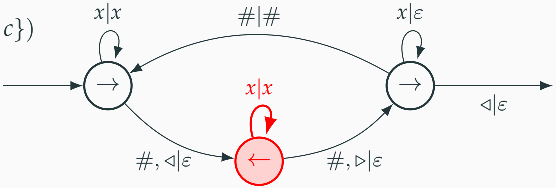
Two-way transducers: executive summary

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

Example:

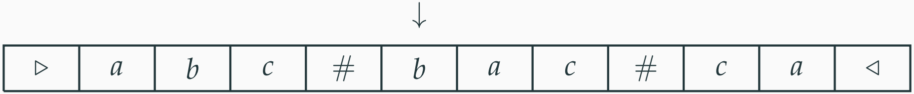
$$\texttt{mapPalin}: \quad \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:

abccba#bacca



Deterministic two-way transducers (2DFT)

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

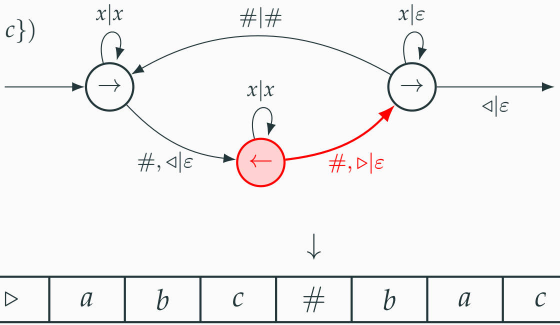
Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin}: \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1\cdot\texttt{reverse}(w_1)\#\ldots\#w_n\cdot\texttt{reverse}(w_n)$$

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



Output:

abccba#baccab

Deterministic two-way transducers (2DFT)

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

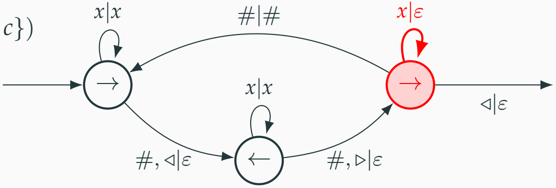
Two-way transducers: executive summary

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

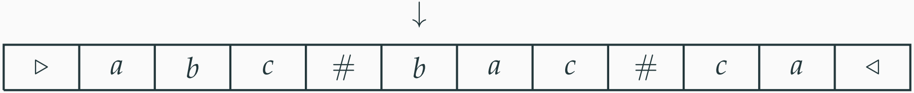
Example:

$$\texttt{mapPalin} : \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:
abccba#baccab



Deterministic two-way transducers (2DFT)

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

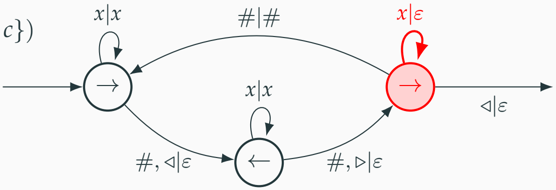
Two-way transducers: executive summary

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

Example:

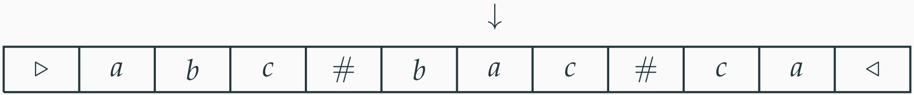
$$\texttt{mapPalin}: \quad \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:

abccba#baccab



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

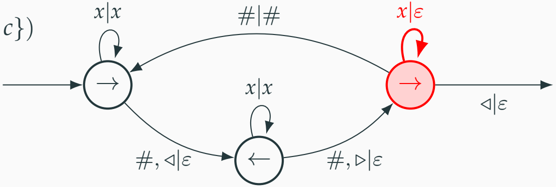
Two-way transducers: executive summary

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

Example:

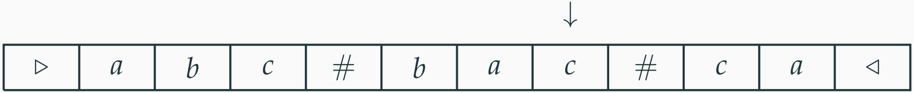
$$\begin{aligned} \texttt{mapPalin} : \quad \{a,b,c,\#\}^* &\longrightarrow \{a,b,c,\#\}^* \\ w_1\#\ldots\#w_n &\longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n) \end{aligned}$$

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



Output:

abccba#baccab



Deterministic two-way transducers (2DFT)

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

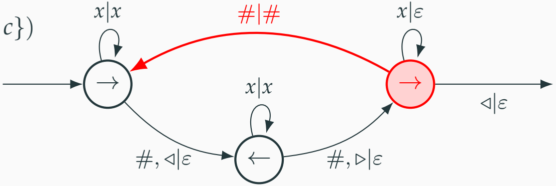
Two-way transducers: executive summary

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

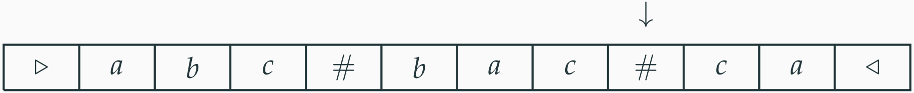
Example:

$$\texttt{mapPalin}: \quad \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:
abccba#baccab



Deterministic two-way transducers (2DFT)

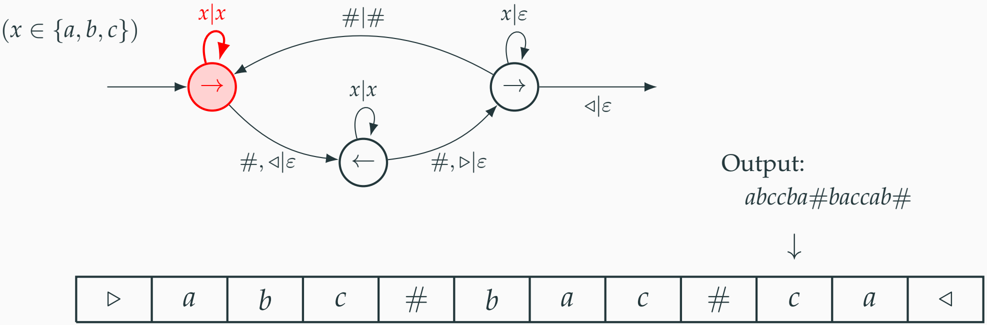
Topic of this talk: certain basic computation models for string-to-string functions

Two-way transducers: executive summary

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

Example:

$$\begin{array}{rccl} \texttt{mapPalin}: & \{a,b,c,\#\}^* & \longrightarrow & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \longmapsto & w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n) \end{array}$$



Deterministic two-way transducers (2DFT)

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

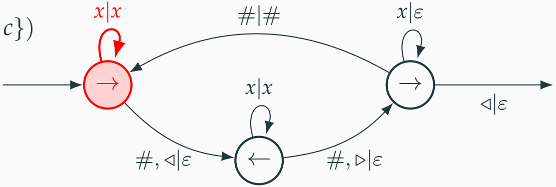
Two-way transducers: executive summary

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

Example:

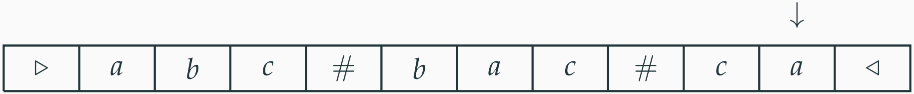
$$\begin{array}{rccc} \texttt{mapPalin}: & \{a,b,c,\#\}^* & \longrightarrow & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \longmapsto & w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n) \end{array}$$

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



Output:

abccba#baccab#c



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

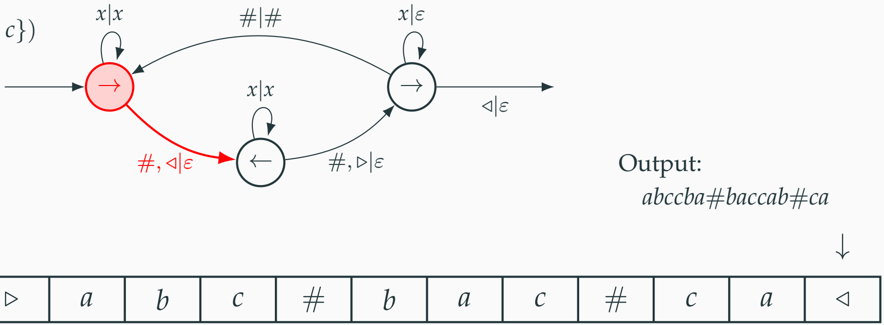
Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin}: \quad \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



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

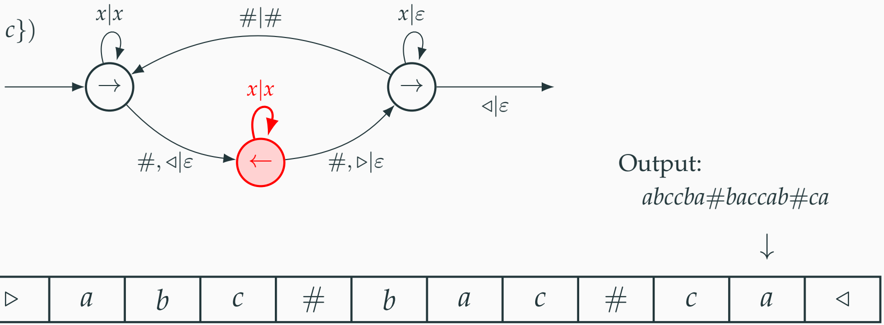
Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin}: \quad \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Deterministic two-way transducers (2DFT)

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

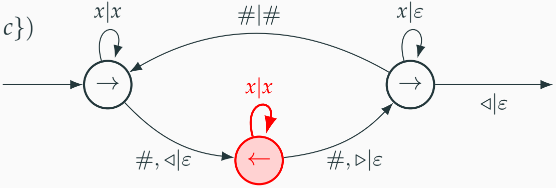
Two-way transducers: executive summary

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

Example:

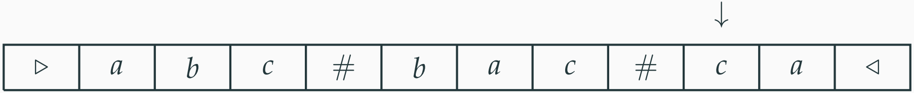
$$\texttt{mapPalin} : \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$$

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



Output:

abccba#baccab#caa



Deterministic two-way transducers (2DFT)

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

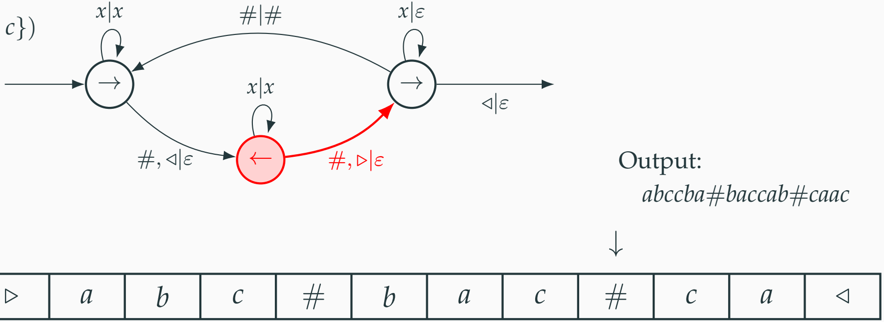
Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin} : \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



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

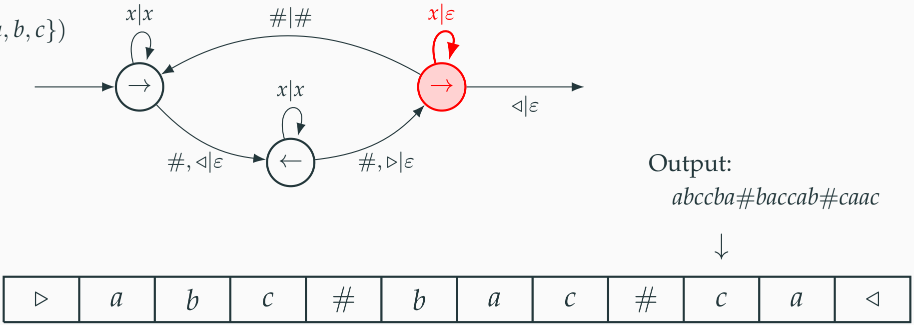
Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin} : \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



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

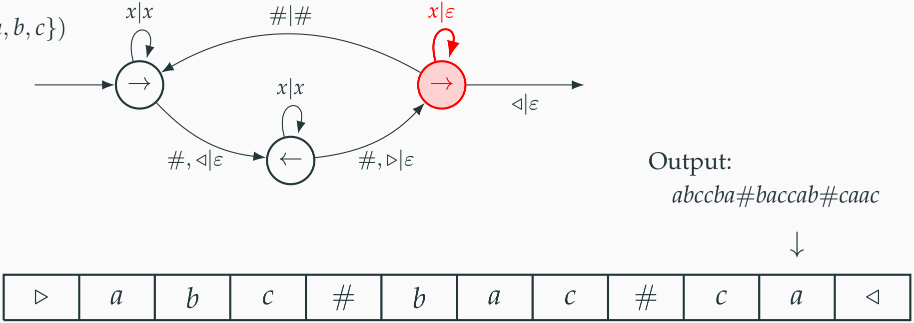
Two-way transducers: executive summary

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

Example:

$$\texttt{mapPalin}: \{a,b,c,\#\}^* \longrightarrow \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$$

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



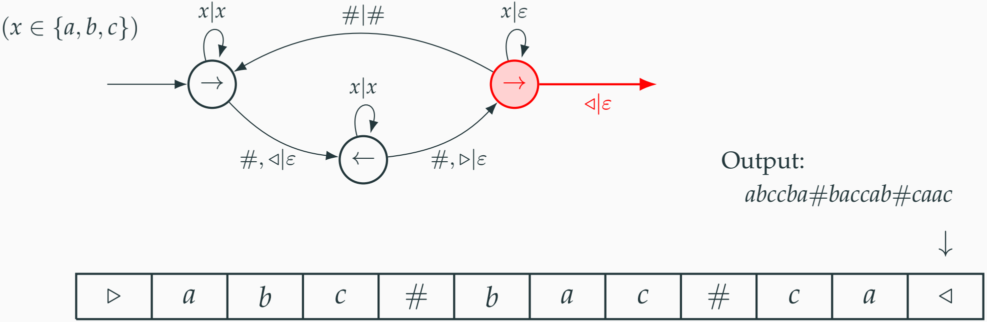
Topic of this talk: certain basic computation models for string-to-string functions

Two-way transducers: executive summary

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

Example:

$$\begin{array}{llll} \texttt{mapPalin}: & \{a,b,c,\#\}^* & \longrightarrow & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \longmapsto & w_1\cdot\texttt{reverse}(w_1)\#\ldots\#w_n\cdot\texttt{reverse}(w_n) \end{array}$$



Regular functions

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

Properties of regular functions

- Linear growth: $|f(w)| = O(|w|)$
- Closed under composition (if $f : \Gamma^* \to \Sigma^*$ and $g : \Sigma^* \to \Pi^*$ 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 λ-calculus and Church encodings [Nguyễn,Noûs,P. 2020]

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 λ-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]

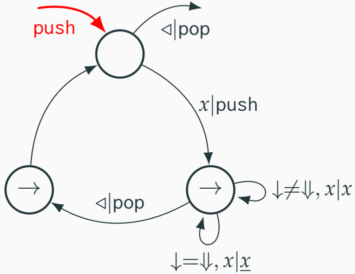
- Multidimensional MSO interpretations
- Imperative nested loop programs
- Simply typed λ-calculus augmented with a list type and some list manipulation primitives
- Composition closure of [regular functions ∪ "squaring with underlining"]
- ***k*-pebble string-to-string transducers**

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" ($k = 2$)



$$\texttt{squaring}: \quad \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$

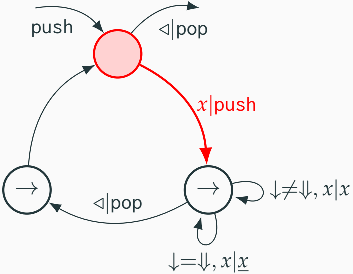
output =

k-pebble transducers: executive summary

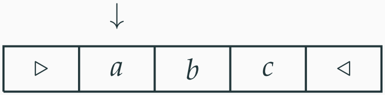
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" ($k = 2$)



$$\texttt{squaring}: \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$



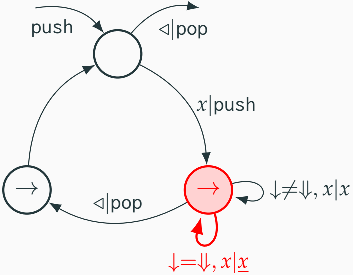
output =

k-pebble transducers: executive summary

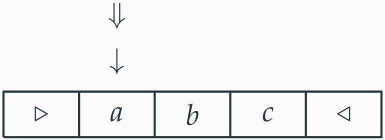
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" $(k = 2)$



$$\begin{array}{rccl} \texttt{squaring}: & \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ & aab & \mapsto & \underline{a}ab\,a\underline{a}b\,aa\underline{b} \end{array}$$



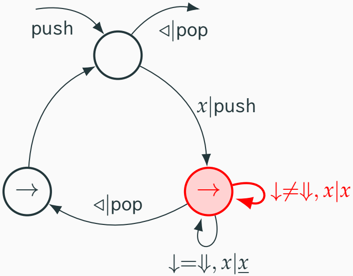
output =

k-pebble transducers: executive summary

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

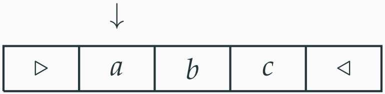
- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers $\cong$ 2DFTs

Example: "squaring with underlining" ($k = 2$)



$$\begin{array}{llll} \texttt{squaring}: & \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ & aab & \mapsto & \underline{a}aba\underline{a}baa\underline{b} \end{array}$$

$\Downarrow$



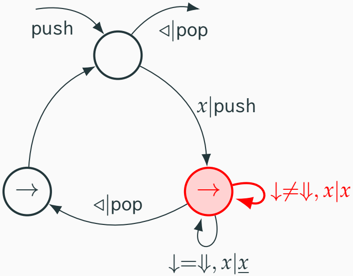
output = $\underline{a}$

k-pebble transducers: executive summary

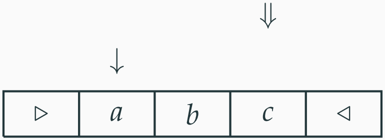
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" ($k = 2$)



$$\texttt{squaring}: \quad \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$



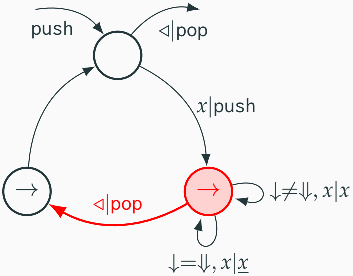
output $= \underline{a}b$

k-pebble transducers: executive summary

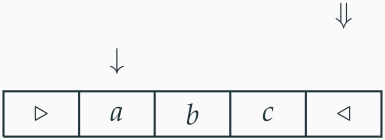
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" $(k = 2)$



$$\texttt{squaring}: \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$



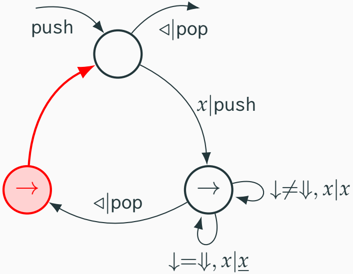
output $= \underline{a}bc$

k-pebble transducers: executive summary

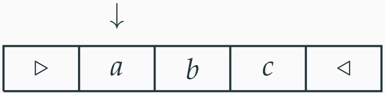
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" ($k = 2$)



$$\texttt{squaring}: \quad \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}ab\,a\underline{a}b\,aa\underline{b}$$



output $= \underline{a}bc$

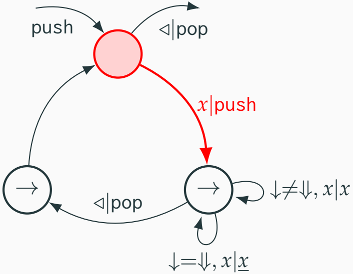
Pebble transducers

k-pebble transducers: executive summary

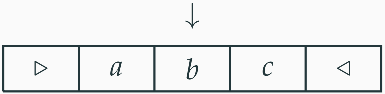
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

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



$$\texttt{squaring}: \quad \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$



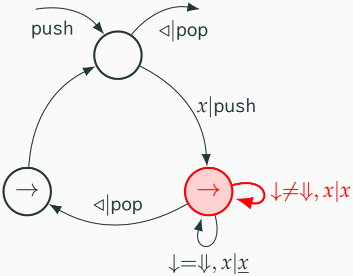
output $= \underline{a}bc$

k-pebble transducers: executive summary

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

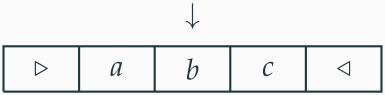
- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers $\cong$ 2DFTs

Example: "squaring with underlining" ($k = 2$)



$$\begin{array}{llll} \texttt{squaring}: & \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ & aab & \mapsto & \underline{a}ab a\underline{a}b aa\underline{b} \end{array}$$

$\Downarrow$



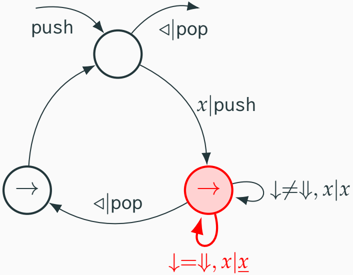
output $= \underline{a}bc$

k-pebble transducers: executive summary

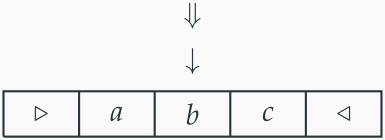
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" ($k = 2$)



$$\texttt{squaring}: \quad \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$



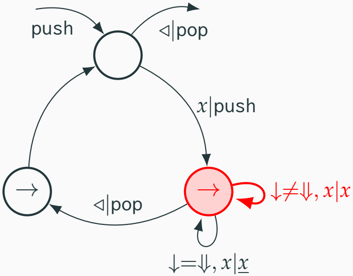
output $= \underline{a}bca$

k-pebble transducers: executive summary

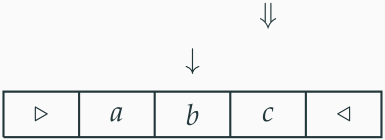
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" ($k = 2$)



$$\texttt{squaring}: \quad \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$



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

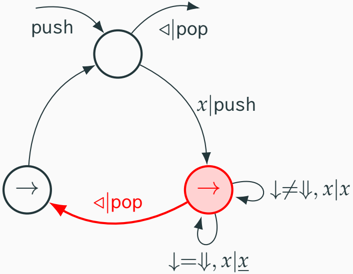
Pebble transducers

k-pebble transducers: executive summary

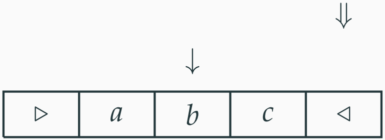
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

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



$$\begin{array}{rccc} \texttt{squaring}: & \Sigma^* & \rightarrow & (\Sigma \cup \underline{\Sigma})^* \\ & aab & \mapsto & \underline{a}aba\underline{a}baa\underline{b} \end{array}$$



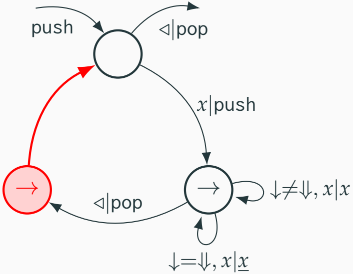
output $= \underline{a}bca\underline{b}c$

k-pebble transducers: executive summary

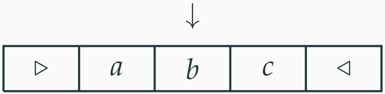
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" ($k = 2$)



$$\texttt{squaring}: \quad \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$



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

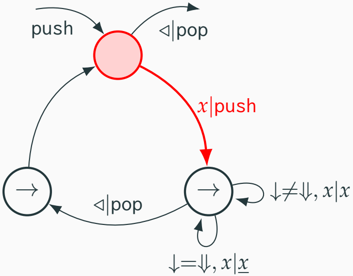
Pebble transducers

k-pebble transducers: executive summary

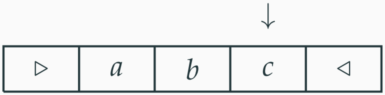
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" ($k = 2$)



$$\texttt{squaring}: \quad \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$



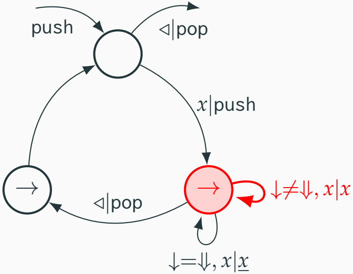
output = $\underline{a}bca\underline{b}c$

k-pebble transducers: executive summary

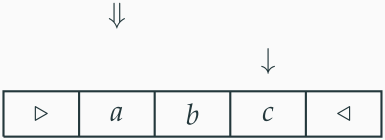
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" ($k = 2$)



$$\texttt{squaring}: \quad \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$



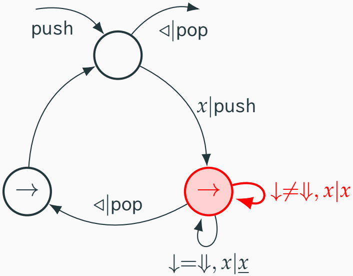
output $= \underline{a}bca\underline{b}c$

k-pebble transducers: executive summary

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

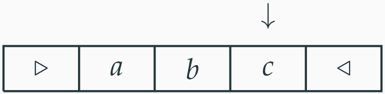
- Heads can be moved, pushed, popped
- Arbitrary comparisons between heads in the stack
- 1-pebble transducers $\cong$ 2DFTs

Example: "squaring with underlining" ($k = 2$)



$$\begin{array}{rccc} \texttt{squaring}: & \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ & aab & \mapsto & \underline{a}aba\underline{a}baa\underline{b} \end{array}$$

$\Downarrow$



output $= \underline{a}bca\underline{b}ca$

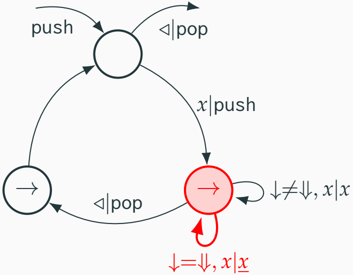
Pebble transducers

k-pebble transducers: executive summary

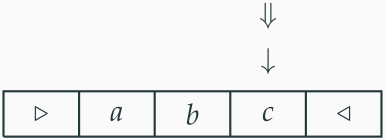
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" ($k = 2$)



$$\texttt{squaring}: \quad \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$



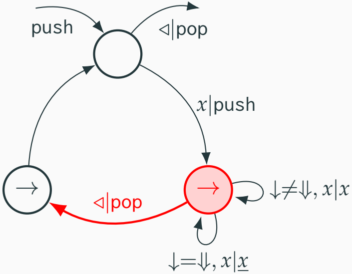
output $= \underline{a}bca\underline{b}cab$

k-pebble transducers: executive summary

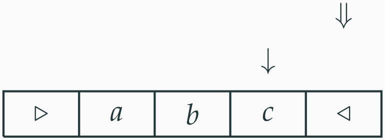
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" ($k = 2$)



$$\texttt{squaring} : \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}ab\,a\underline{a}b\,aa\underline{b}$$



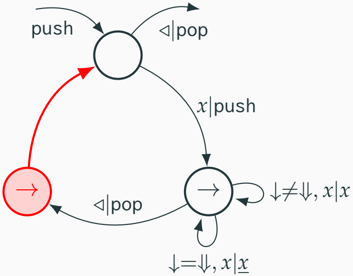
output $= \underline{a}bca\underline{b}cab\underline{c}$

k-pebble transducers: executive summary

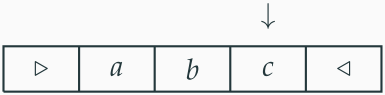
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

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



$$\texttt{squaring} : \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}aba\underline{a}baa\underline{b}$$



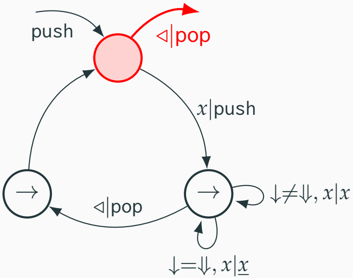
output $= \underline{a}bca\underline{b}cab\underline{c}$

k-pebble transducers: executive summary

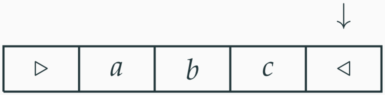
Finite set of states + a stack of two-way reading heads of height $\leq k$

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

Example: "squaring with underlining" ($k = 2$)



$$\texttt{squaring}: \quad \Sigma^* \to (\Sigma \cup \underline{\Sigma})^*$$
$$aab \mapsto \underline{a}ab\,a\underline{a}b\,aa\underline{b}$$



output $= \underline{a}bc\,a\underline{b}c\,ab\underline{c}$

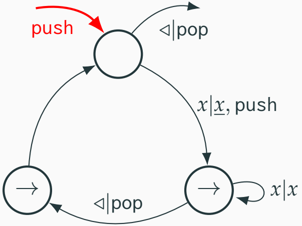
Comparison-free pebble transducers

Main question

What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring" $\texttt{cfsquaring}(abb) = \underline{a}abb\underline{b}abb\underline{b}abb$



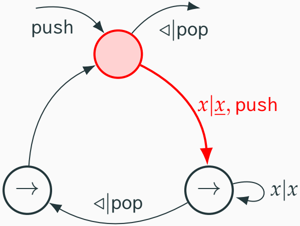
output =

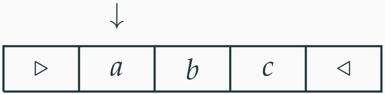
Main question

What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring" $\texttt{cfsquaring}(abb) = \underline{a}abb\underline{b}abb\underline{b}abb$





output =

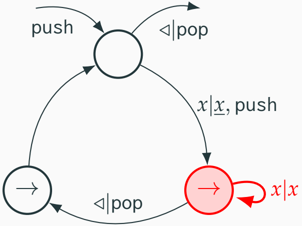
Main question

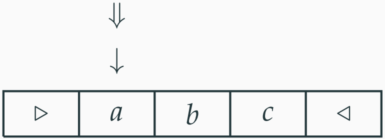
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”

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





output $= \underline{a}$

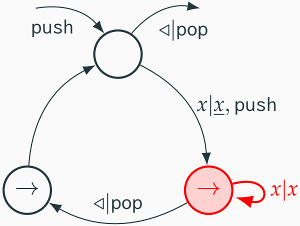
Main question

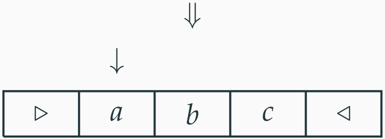
What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring"

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





output $= \underline{a}a$

Main question

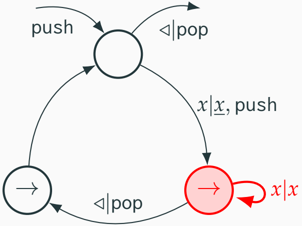
What happens if we disallow comparisons between reading heads?

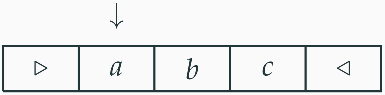
Non-example: "squaring with underlining"

Example: "comparison-free squaring"

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

$\Downarrow$





output $= \underline{a}ab$

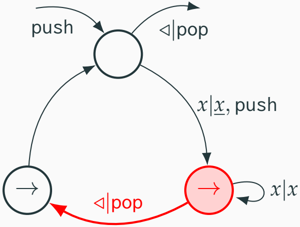
Main question

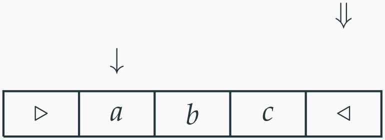
What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring"

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





output $= \underline{a}abc$

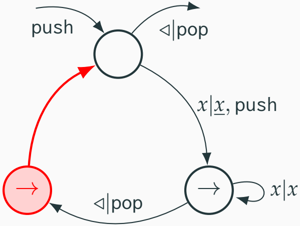
Main question

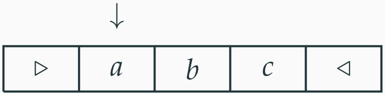
What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring"

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





output $= \underline{a}abc$

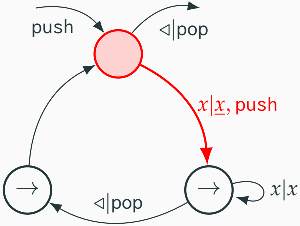
Main question

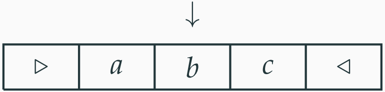
What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring"

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





output $= \underline{a}abc$

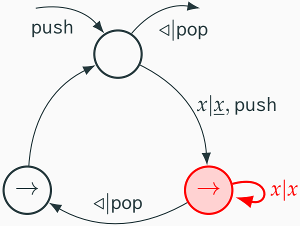
Main question

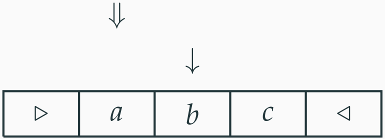
What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring"

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





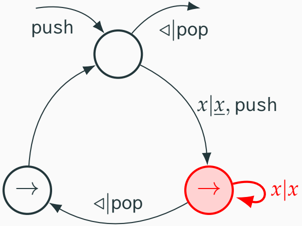
output $= \underline{a}abc\underline{b}$

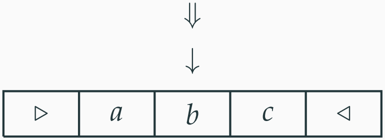
Main question

What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring" $\texttt{cfsquaring}(abb) = \underline{a}abb\underline{b}abb\underline{b}abb$





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

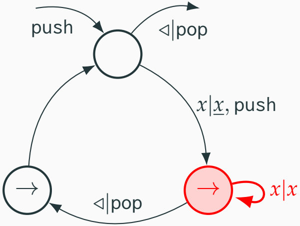
Main question

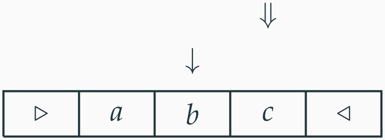
What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring"

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





output $= \underline{a}abc\underline{b}ab$

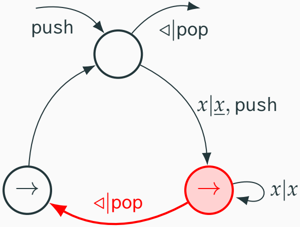
Main question

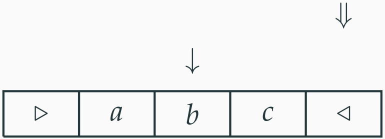
What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring"

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





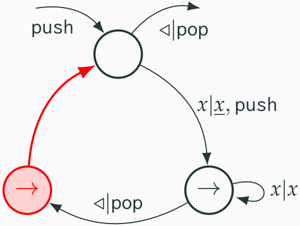
output $= \underline{a}abc\underline{b}abc$

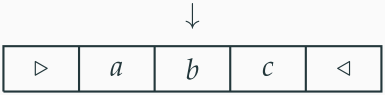
Main question

What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring" $\texttt{cfsquaring}(abb) = \underline{a}abb\underline{b}abb\underline{b}abb$





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

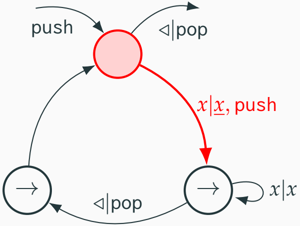
Main question

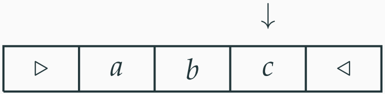
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”

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





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

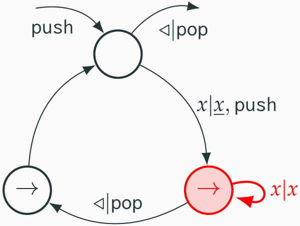
Main question

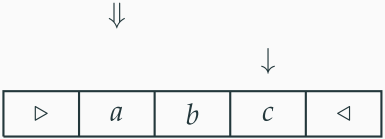
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”

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





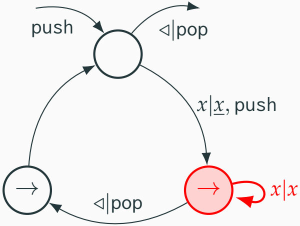
output $= \underline{a}abc\underline{b}abc\underline{c}$

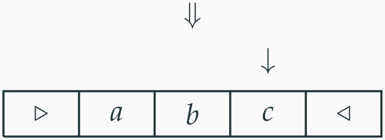
Main question

What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring" $\texttt{cfsquaring}(abb) = \underline{a}abb\underline{b}abb\underline{b}abb$





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

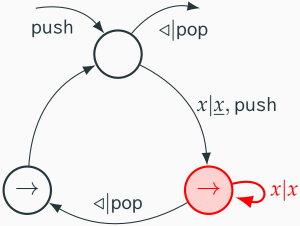
Main question

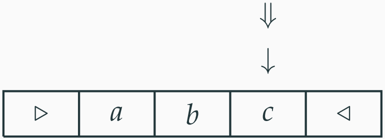
What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring"

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





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

Comparison-free pebble transducers

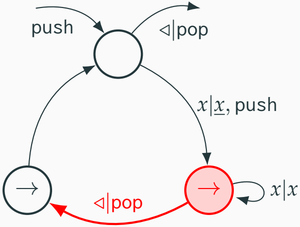
Main question

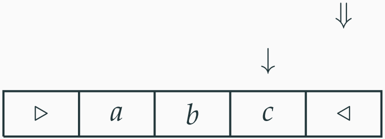
What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring"

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





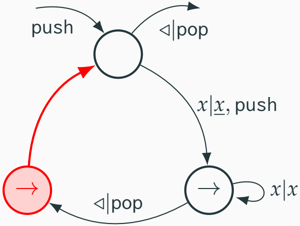
output $= \underline{a}abc\underline{b}abc\underline{c}abc$

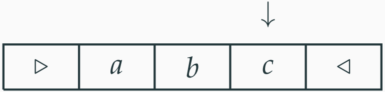
Main question

What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring” $\texttt{cfsquaring}(abb) = \underline{a}abb\underline{b}abb\underline{b}abb$





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

Comparison-free pebble transducers

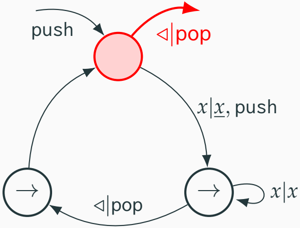
Main question

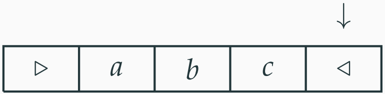
What happens if we disallow comparisons between reading heads?

Non-example: “squaring with underlining”

Example: “comparison-free squaring”

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





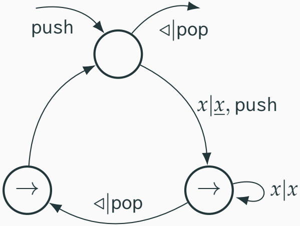
output $= \underline{a}abc\underline{b}abc\underline{c}abc$

Main question

What happens if we disallow comparisons between reading heads?

Non-example: "squaring with underlining"

Example: "comparison-free squaring" $\texttt{cfsquaring}(abb) = \underline{a}abb\underline{b}abb\underline{b}abb$



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

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 Γ, Σ and I be finite alphabets and $f : \Gamma^* \to I^*$, $g_i : \Gamma^* \to \Sigma^*$ and $w \in \Gamma^*$.
Define $\mathrm{CbS}(f, (g_i)_{i \in I})(w)$ so that, if $f(w) = i_1 \dots i_k$, then

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

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

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

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

Alternative characterization (1/2): composition by substitutions

Definition (Composition by substitutions)

Let Γ, Σ and I be finite alphabets and $f : \Gamma^* \to I^*$, $g_i : \Gamma^* \to \Sigma^*$ and $w \in \Gamma^*$.
Define $\mathrm{CbS}(f, (g_i)_{i \in I})(w)$ so that, if $f(w) = i_1 \dots i_k$, then

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

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

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

- 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 $\mathrm{CbS}(f, (g_i)_{i \in I})$ is cfpolyreg

Alternative characterization (1/2): composition by substitutions

Definition (Composition by substitutions)

Let Γ, Σ and I be finite alphabets and $f : \Gamma^* \to I^*$, $g_i : \Gamma^* \to \Sigma^*$ and $w \in \Gamma^*$.
Define $\mathrm{CbS}(f, (g_i)_{i \in I})(w)$ so that, if $f(w) = i_1 \dots i_k$, then

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

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

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

- 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 $\mathrm{CbS}(f, (g_i)_{i \in I})$ is cfpolyreg

- More convenient to manipulate formally
- Tight link between the number of pebbles and the nesting of the CbS operator

Alternative characterization (2/2): composition and basic combinators

We have an alternative characterization based on linear the λ-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 λ-calculus)

Closure under composition

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

Alternative characterization (2/2): composition and basic combinators

We have an alternative characterization based on linear the λ-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 λ-calculus)

Closure under composition

If $f : \Sigma^* \to \Gamma^*$ and $g : \Gamma^* \to \Delta^*$ 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 $\to$ all polyregular functions

- Regular functions can also themselves be decomposed

Not all polyregular functions are comparison-free

Separation results

Theorem

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

Corollary: "squaring with underlining" is not CF.

Theorem

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

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

Separation results

Theorem

The function $f : a^n \in \{a\}^* \mapsto a\#aa\# \ldots \#a^n$ 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

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

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

Separation results

Theorem

The function $f : a^n \in \{a\}^* \mapsto a\#aa\#\ldots\#a^n$ 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

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

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

Definition

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

$g = \mathbf{map}(w \mapsto w^{|w|})$ therefore comparison-free polyregular functions are *not* closed under **map**, unlike regular and polyreg functions

→ obstruction to characterizing cfpolyreg fn by list-processing functional programs (à la [Bojańczyk, Daviaud & Krishna 2018])

Separation proof idea for "map unary square"

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,

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

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

$g(a^{n_1}\# \ldots \#a^{n_k}) = a^{n_1 \times n_1}\# \ldots \#a^{n_k \times n_k}$ *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,

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

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

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

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

$$a \quad aa \quad \ldots \quad a^n$$

Lemma

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

Separation proof idea continued: unary inputs

Theorem

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

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

$$a \quad aa \quad \ldots \quad a^n$$

Lemma

For any cfpolyreg $g : \{a\}^* \to \Sigma^*$, 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 $\{$lengths of maximal a-factors of $g(a^n)\}$, by structural induction on *poly-pumping sequences*.

Separation proof idea continued: unary inputs

Theorem

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

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

$$a \quad aa \quad \ldots \quad a^n$$

Lemma

For any cfpolyreg $g : \{a\}^* \to \Sigma^*$, 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 $\{$lengths of maximal a-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

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

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

$$a \quad aa \quad \ldots \quad a^n$$

Lemma

For any cfpolyreg $g : \{a\}^* \to \Sigma^*$, 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 {lengths of maximal a-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)

$f : \{a\}^* \to \Sigma^*$ 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

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

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

$$a \quad aa \quad \ldots \quad a^n$$

Lemma

For any cfpolyreg $g : \{a\}^* \to \Sigma^*$, 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 {lengths of maximal a-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)

$f : \{a\}^* \to \Sigma^*$ 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.)

- Regular word sequences are u.p.c. of pumping sequences $(u_0(v_1)^n \ldots (v_l)^n u_l)_{n\in\mathbb{N}}$ [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

Further topics

Going first-order

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]

$\rightsquigarrow$ Analogous class of FO-polyregular.

Going first-order

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]

$\rightsquigarrow$ Analogous class of FO-polyregular. What about cfpolyregular functions?

Going first-order

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]

$\rightsquigarrow$ 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 $\mathrm{CbS}(f, (g_i)_{i \in I})$ is FO-cfpolyreg

Going first-order

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]

$\rightsquigarrow$ 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 $\mathrm{CbS}(f, (g_i)_{i \in I})$ is FO-cfpolyreg

Other characterizations?

Logical characterization of FO-cfpolyregular functions

Let $\mathfrak{M} : \{\text{words}\} \to \{\text{finite models}\}$ be as usual.
For $\mathfrak{U} = (U, R, \ldots)$, let $\mathfrak{U}^k = (U^k, R_1, \ldots, R_k, \ldots)$ where $R_i(x_1, \ldots, x_m) :\Leftrightarrow R(\pi_i(x_1), \ldots, \pi_i(x_m))$.

Conjecture

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

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

Logical characterization of FO-cfpolyregular functions

Let $\mathfrak{M} : \{\text{words}\} \to \{\text{finite models}\}$ be as usual.
For $\mathfrak{U} = (U, R, \ldots)$, let $\mathfrak{U}^k = (U^k, R_1, \ldots, R_k, \ldots)$ where $R_i(x_1, \ldots, x_m) :\Leftrightarrow R(\pi_i(x_1), \ldots, \pi_i(x_m))$.

Conjecture

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

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

Logical characterization of FO-cfpolyregular functions

Let $\mathfrak{M} : \{\text{words}\} \to \{\text{finite models}\}$ be as usual.
For $\mathfrak{U} = (U, R, \ldots)$, let $\mathfrak{U}^k = (U^k, R_1, \ldots, R_k, \ldots)$ where $R_i(x_1, \ldots, x_m) :\Leftrightarrow R(\pi_i(x_1), \ldots, \pi_i(x_m))$.

Conjecture

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

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

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

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

Further topics

A few relevant directions:

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

  A linear λ-calculus characterization matches an analogue of the CbS definition

Further topics

A few relevant directions:

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

  A linear λ-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

Further topics

A few relevant directions:

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

  A linear λ-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 λ-calculus characterization for the FO case

Conclusion

Summary

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 +** `cfsquaring`$(abc) = \underline{a}abc\underline{b}abc\underline{c}abc$

- 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
  - $a^n \mapsto a\#aa\#\ldots\#a^n$ and "map unary square" are polyregular but not cfpolyreg
  - for $\{a\}^* \to \{a\}^*$ cfpolyreg = polyreg
- Incomparable with polynomial HDT0L transductions
  - $a^n \mapsto a\#aa\#\ldots\#a^n$ not cfpolyreg but HDT0L
  - $w \mapsto w^{|w|}$ is cfpolyreg but not HDT0L
- Well-behaved first-order counterpart

Summary

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 +** `cfsquaring`$(abc) = \underline{a}abc\underline{b}abc\underline{c}abc$

- 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
  - $a^n \mapsto a\#aa\#\ldots\#a^n$ and "map unary square" are polyregular but not cfpolyreg
  - for $\{a\}^* \to \{a\}^*$ cfpolyreg = polyreg
- Incomparable with polynomial HDT0L transductions
  - $a^n \mapsto a\#aa\#\ldots\#a^n$ 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