Algebraic Recognition of Regular Functions

Lê Thành Dũng (Tito) Nguyễn — `nltd@nguyentito.eu` — ÉNS Lyon
joint work with Mikołaj Bojańczyk (MIMUW, University of Warsaw)

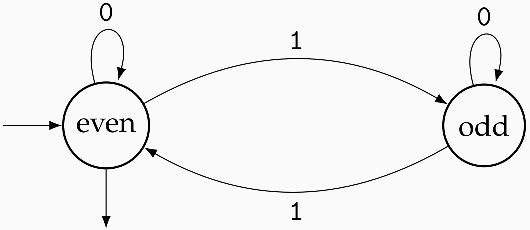
11th **Sy**mposium on **Co**mpositional Structures – April 20th, 2023

Reminder: automata and regular languages

Languages = sets of words $L \subseteq \Sigma^* \cong$ decision problems $\Sigma^* \to \{\text{yes}, \text{no}\}$

Regular languages: fundamental class in comp. sci., many definitions

- *regular expressions*: `0*(10*10*)*` = "only `0`s and `1`s & even number of `1`s"
- *finite automata* (deterministic or not): e.g. drawing below



Reminder: automata and regular languages

Languages = sets of words $L \subseteq \Sigma^*$ $\cong$ decision problems $\Sigma^* \to \{\text{yes}, \text{no}\}$

Regular languages: fundamental class in comp. sci., many definitions

- *regular expressions*: `0*(10*10*)*` = "only `0`s and `1`s & even number of `1`s"
- *finite automata* (deterministic or not)
- *algebraic* definition below (very close to automata), e.g. $M = \mathbb{Z}/(2)$

Theorem (classical)

A language $L \subseteq \Sigma^*$ *is* regular $\iff$ *there are a* monoid morphism $\varphi\colon \Sigma^* \to M$ *to a* finite monoid M *and a subset* $P \subseteq M$ *such that* $L = \varphi^{-1}(P) = \{w \in \Sigma^* \mid \varphi(w) \in P\}$.

$\Sigma^* = \{\text{words over the finite alphabet } \Sigma\} =$ *free monoid*

- monadic 2nd-order logic, simply typed λ-calculus [Hillebrand & Kanellakis 1996], …

Algebraic recognition of regular languages

A language $L \subseteq \Sigma^*$ is regular $\iff$ the corresponding decision problem *factors* as

$$\Sigma^* \xrightarrow{\text{some } \textit{morphism}} \text{some finite monoid } M \to \{\text{yes}, \text{no}\}$$

$\rightsquigarrow$ terminology: *"M recognizes L"*

Algebraic recognition of regular languages

A language $L \subseteq \Sigma^*$ is regular $\iff$ the corresponding decision problem *factors* as

$$\Sigma^* \xrightarrow{\text{some } \textit{morphism}} \text{some finite monoid } M \to \{\text{yes}, \text{no}\}$$

$\rightsquigarrow$ terminology: "M recognizes L"

Varying the monoids M allowed leads to *algebraic language theory*

Founding example: Schützenberger's theorem on star-free languages

L is recognized by some *aperiodic* finite monoid $(\forall x \in M,\ \exists n \in \mathbb{N} : x^n = x^{n+1})$

$\iff$ it is described by some *star-free expression*

$$E, E' ::= \varnothing \mid \overbrace{\varepsilon}^{\text{empty string}} \mid \underbrace{a}_{\text{letter in a finite alphabet } \Sigma} \mid E \cup E' \mid \overbrace{E \cdot E'}^{\text{concatenation}} \mid \underbrace{\neg E}_{\text{complement}} \qquad \rightsquigarrow \qquad [\![E]\!] \subseteq \Sigma^*$$

Semigroups instead of monoids

A language $L \subseteq \Sigma^*$ is regular $\iff$ the corresponding decision problem factors as

$$\Sigma^* \xrightarrow{\text{some } \textit{morphism}} \text{some finite } \textit{semigroup } S \to \{\text{yes}, \text{no}\}$$

Definition

Semigroup = set + associative binary operation (so monoid = semigroup + unit)

Semigroups instead of monoids

A language $L \subseteq \Sigma^*$ is regular $\iff$ the corresponding decision problem factors as

$$\Sigma^* \xrightarrow{\text{some } \textit{morphism}} \text{some finite } \textit{semigroup } S \to \{\text{yes}, \text{no}\}$$

Definition

Semigroup = set + associative binary operation (so monoid = semigroup + unit)

We still have: star-free language $\iff$ recognized by *aperiodic* finite semigroup

Semigroups are sometimes more convenient than monoids

A finite semigroup is aperiodic ($\forall x \in S,\ \exists n \geq 1 : x^n = x^{n+1}$)

$\Leftrightarrow$ none of its non-trivial subsemigroups are groups ((⇐) fails with submonoids)

Remark: every finite semigroup $\underbrace{\text{“is built from”}}_{\text{divides a wreath product of}}$ groups & aperiodic semigroups (Krohn–Rhodes decomposition)

From languages to functions

Finite semigroups recognize regular *languages* $L \subseteq \Sigma^*$ $\rightsquigarrow$ leads to a rich theory

What about functions $f : \Sigma^* \to \Gamma^*$?

From languages to functions

Finite semigroups recognize regular *languages* $L \subseteq \Sigma^*$ $\rightsquigarrow$ leads to a rich theory

What about functions $f : \Sigma^* \to \Gamma^*$?

Many non-equivalent *transducer* models: finite-state devices with outputs

(sequential functions, rational functions, polyregular functions...)

common property ("sanity check"): L regular $\implies f^{-1}(L)$ regular

From languages to functions

Finite semigroups recognize regular *languages* $L \subseteq \Sigma^*$ $\rightsquigarrow$ leads to a rich theory

What about <u>functions</u> $f : \Sigma^* \to \Gamma^*$?

Many non-equivalent *transducer* models: finite-state devices with outputs

(sequential functions, rational functions, polyregular functions...)

common property ("sanity check"): L regular $\implies f^{-1}(L)$ regular

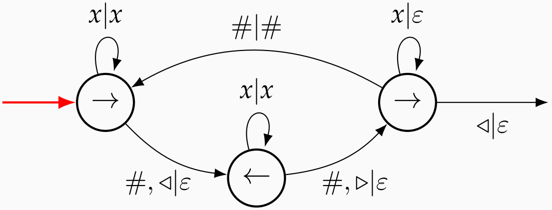
Regular functions are one of the most robust/canonical classes

- several equivalent definitions (next slides)
- previously, no concise algebraic one $\longrightarrow$ **<u>our contribution</u>**

using a bit of category theory!

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \mathtt{reverse}(w_1) \# \dots \# w_n \cdot \mathtt{reverse}(w_n)$



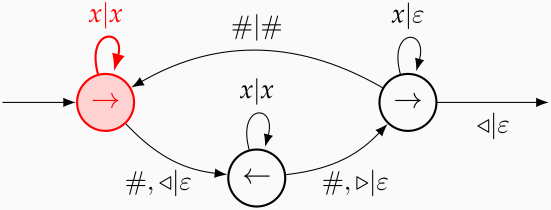
$(x \in \{a, b, c\})$

▷	a	b	c	#	b	a	c	#	c	b	◁

Output:

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$



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

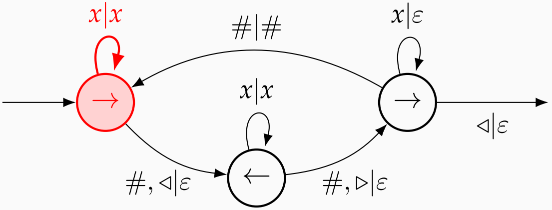
$\downarrow$

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

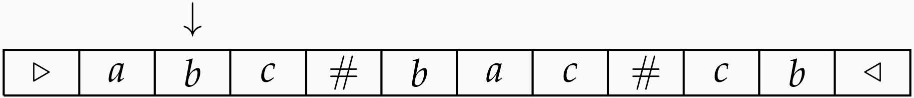
Output:

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



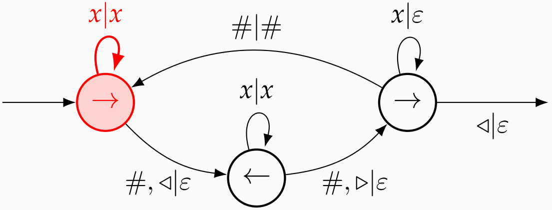
$(x \in \{a, b, c\})$



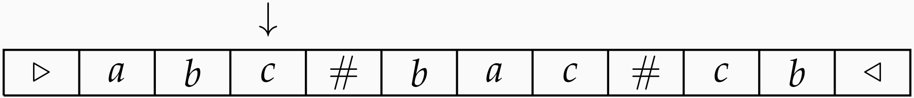
Output: a

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



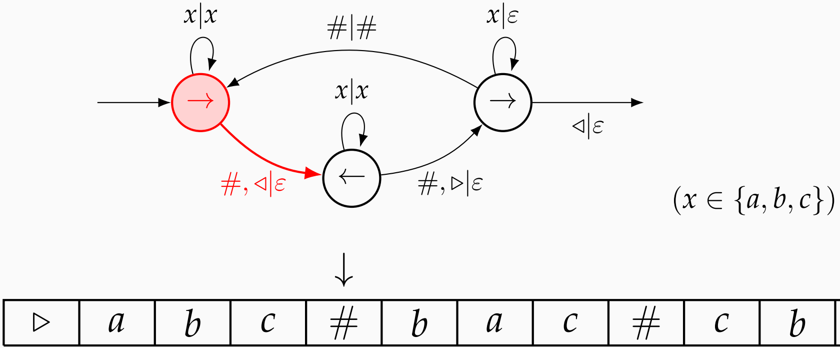
$(x \in \{a, b, c\})$



Output: ab

The first definition of regular functions: (deterministic) two-way transducers

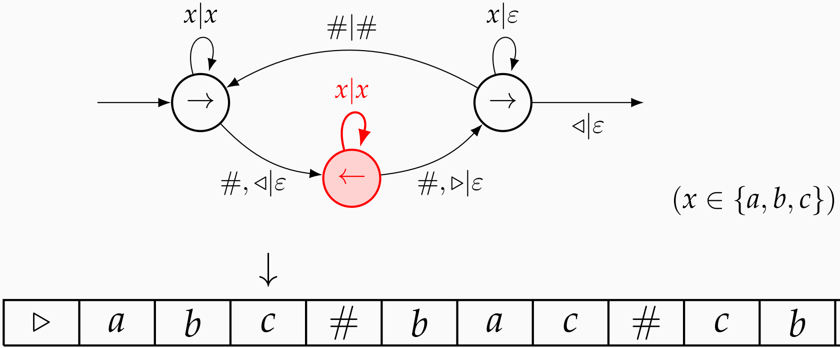
Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$



Output: *abc*

The first definition of regular functions: (deterministic) two-way transducers

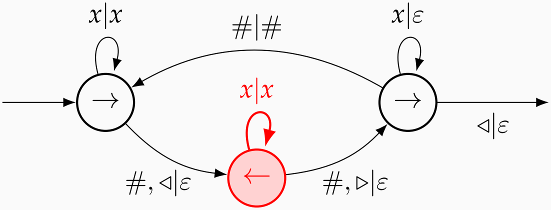
Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \mathtt{reverse}(w_1)\#\ldots\#w_n \cdot \mathtt{reverse}(w_n)$



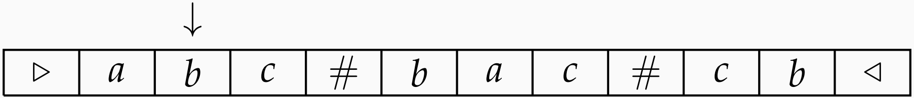
Output: abc

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$



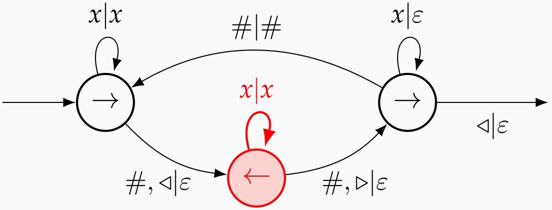
$(x \in \{a, b, c\})$



Output: $abcc$

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \dots \# w_n \cdot \texttt{reverse}(w_n)$



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

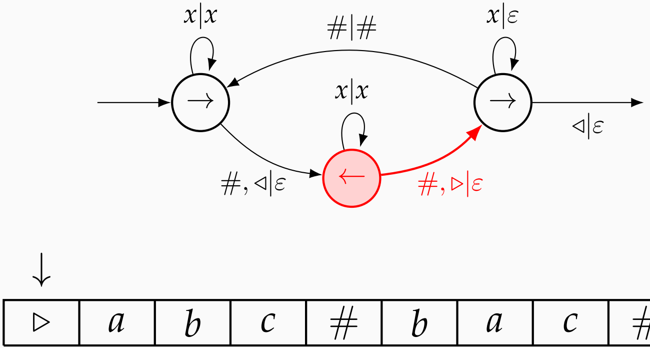
$\downarrow$

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

Output: *abccb*

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\# \dots \#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \dots \#w_n \cdot \texttt{reverse}(w_n)$

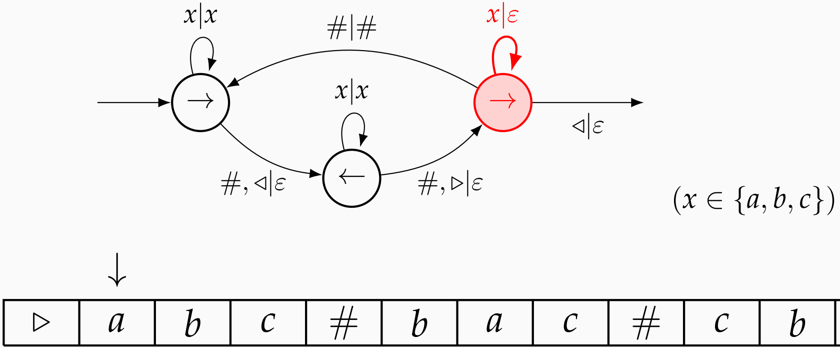


$(x \in \{a, b, c\})$

Output: *abccba*

The first definition of regular functions: (deterministic) two-way transducers

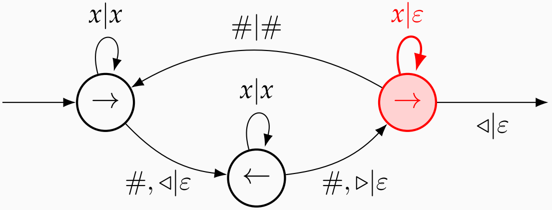
Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \mathtt{reverse}(w_1)\#\ldots\#w_n \cdot \mathtt{reverse}(w_n)$



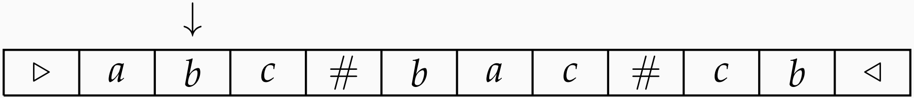
Output: *abccba*

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



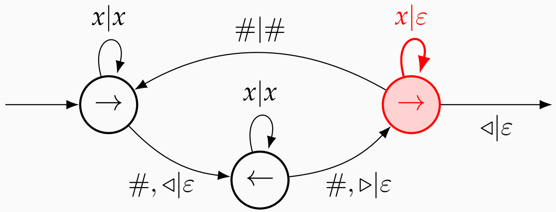
$(x \in \{a, b, c\})$



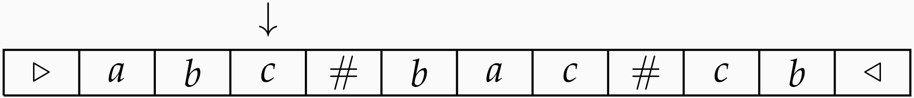
Output: *abccba*

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\# \ldots \# w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \ldots \# w_n \cdot \texttt{reverse}(w_n)$



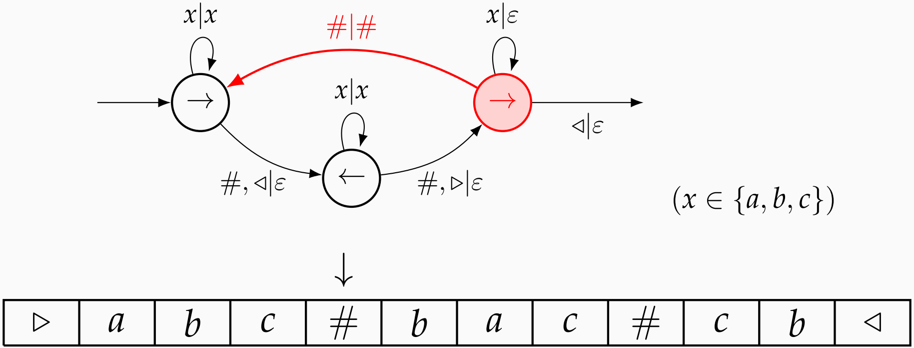
$(x \in \{a, b, c\})$



Output: *abccba*

The first definition of regular functions: (deterministic) two-way transducers

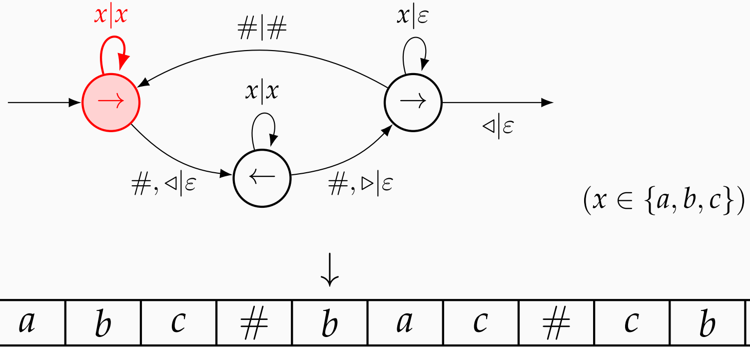
Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



Output: *abccba*

The first definition of regular functions: (deterministic) two-way transducers

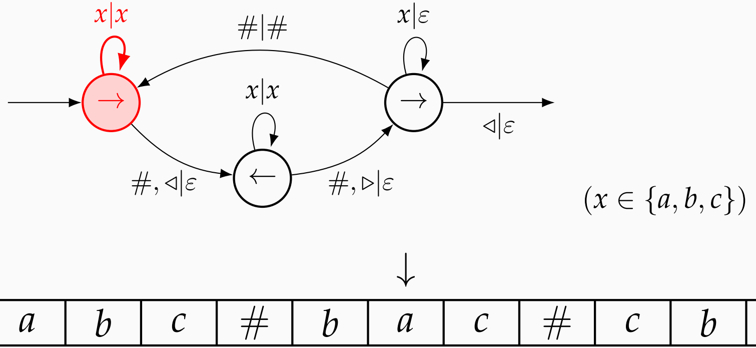
Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \dots \# w_n \cdot \texttt{reverse}(w_n)$



Output: $abccba\#$

The first definition of regular functions: (deterministic) two-way transducers

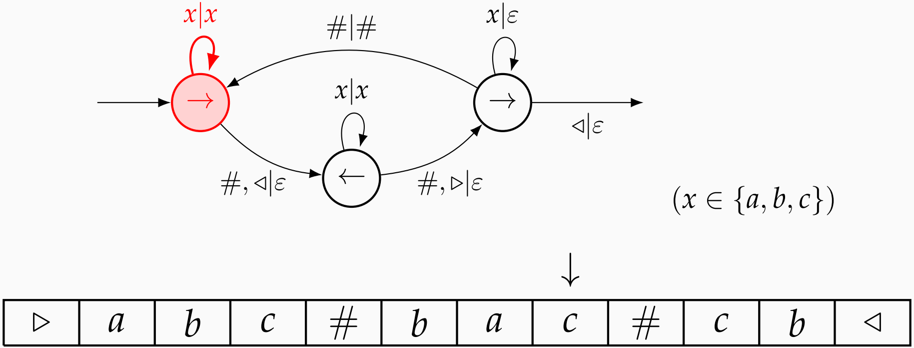
Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$



Output: $abccba\#b$

The first definition of regular functions: (deterministic) two-way transducers

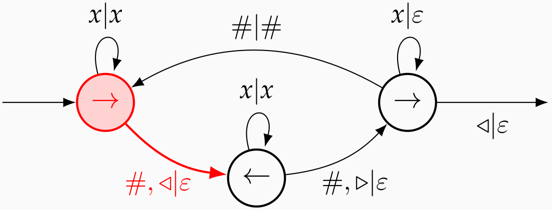
Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \dots \# w_n \cdot \texttt{reverse}(w_n)$



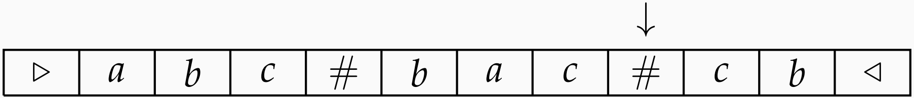
Output: $abccba\#ba$

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$



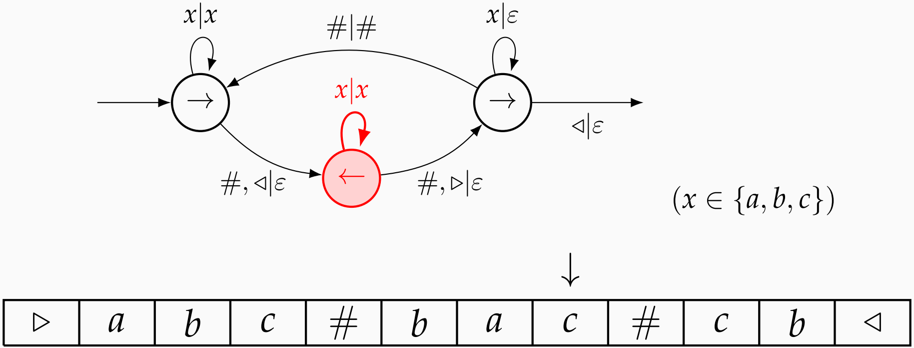
$(x \in \{a, b, c\})$



Output: $abccba\#bac$

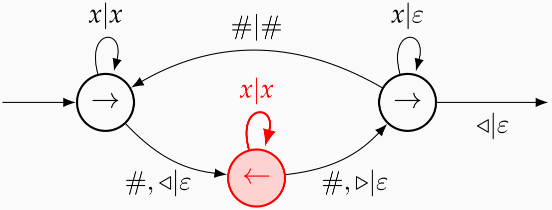
The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \mathtt{reverse}(w_1)\#\dots\#w_n \cdot \mathtt{reverse}(w_n)$

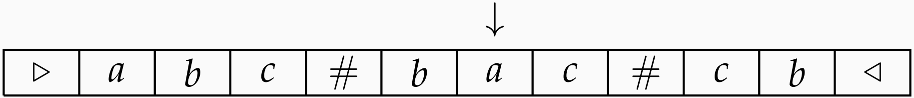


Output: *abccba#bac*

Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \dots \# w_n \cdot \texttt{reverse}(w_n)$



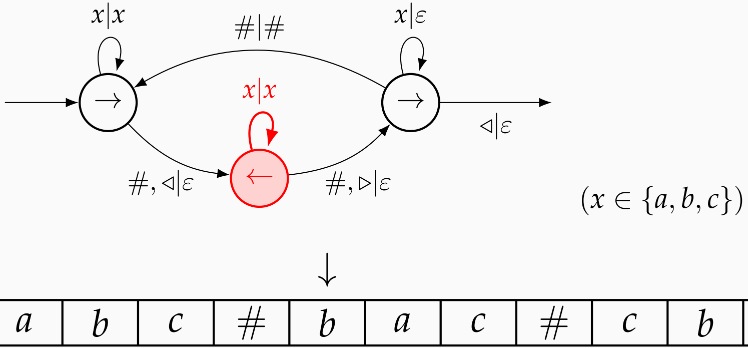
$(x \in \{a, b, c\})$



Output: $abccba\#bacc$

The first definition of regular functions: (deterministic) two-way transducers

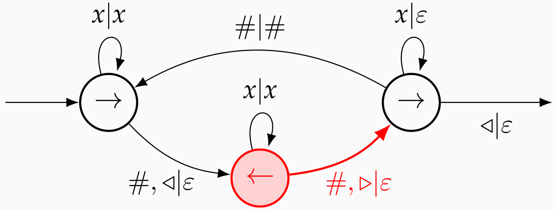
Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \mathtt{reverse}(w_1)\#\ldots\#w_n \cdot \mathtt{reverse}(w_n)$



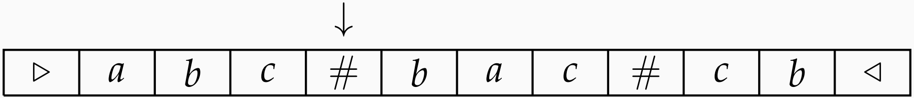
Output: *abccba#bacca*

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



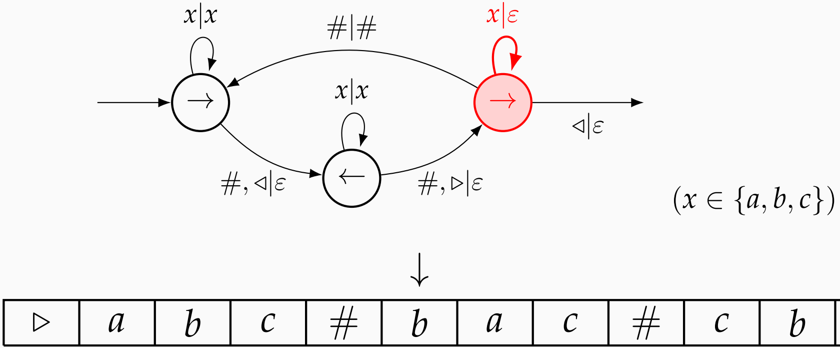
$(x \in \{a, b, c\})$



Output: *abccba#baccab*

The first definition of regular functions: (deterministic) two-way transducers

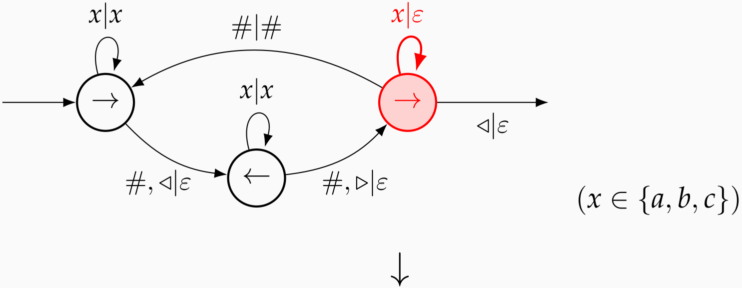
Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



Output: $abccba\#baccab$

The first definition of regular functions: (deterministic) two-way transducers

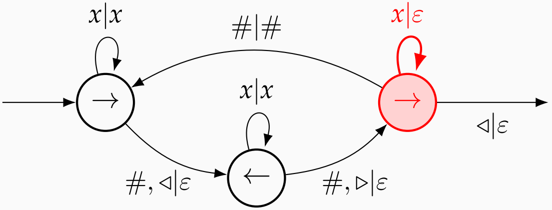
Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



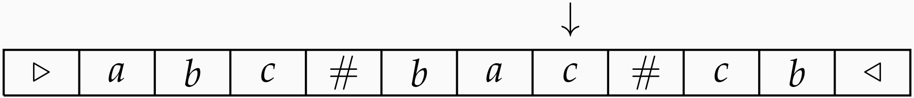
$\triangleright$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\triangleleft$

Output: $abccba\#baccab$

Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



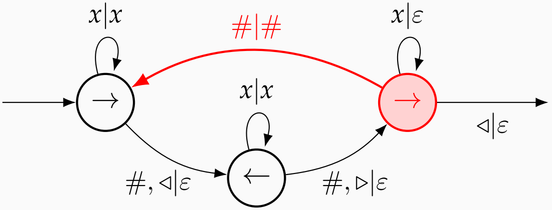
$(x \in \{a, b, c\})$



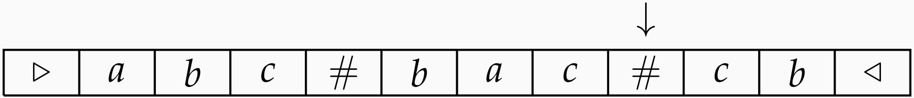
Output: *abccba#baccab*

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \dots \# w_n \cdot \texttt{reverse}(w_n)$



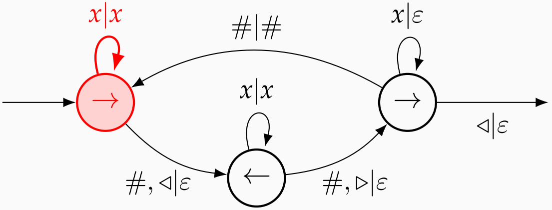
$(x \in \{a, b, c\})$



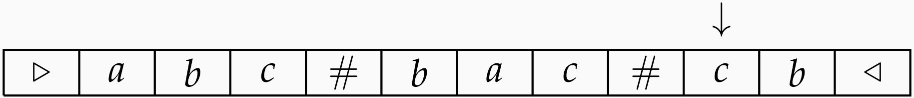
Output: *abccba#baccab*

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \mathtt{reverse}(w_1)\# \dots \# w_n \cdot \mathtt{reverse}(w_n)$



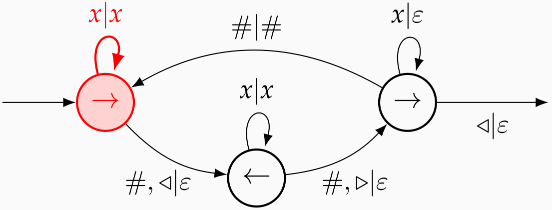
$(x \in \{a, b, c\})$



Output: $abccba\#baccab\#$

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\# \ldots \# w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \ldots \# w_n \cdot \texttt{reverse}(w_n)$



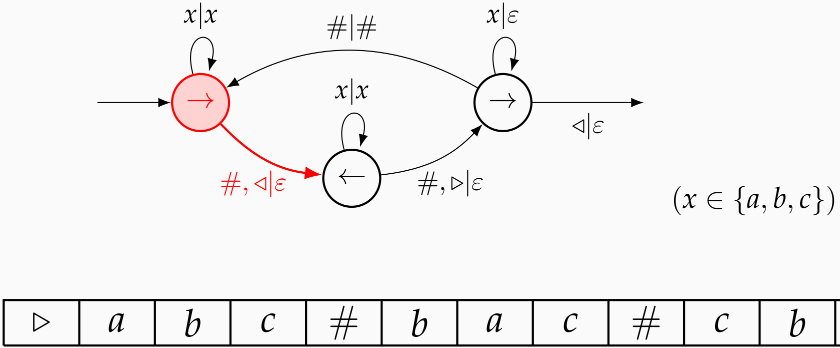
$(x \in \{a, b, c\})$

▷	a	b	c	#	b	a	c	#	c	b ↓	◁

Output: $abccba\#baccab\#c$

The first definition of regular functions: (deterministic) two-way transducers

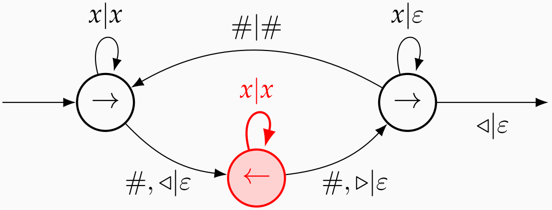
Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \dots \# w_n \cdot \texttt{reverse}(w_n)$



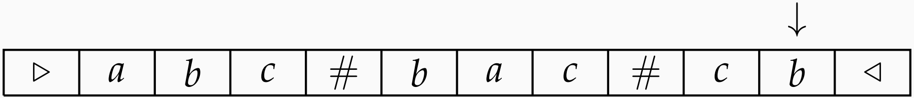
Output: $abccba\#baccab\#cb$

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



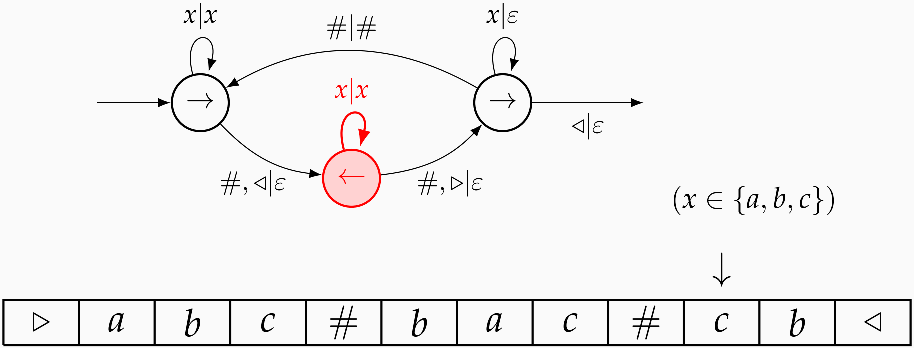
$(x \in \{a, b, c\})$



Output: $abccba\#baccab\#cb$

The first definition of regular functions: (deterministic) two-way transducers

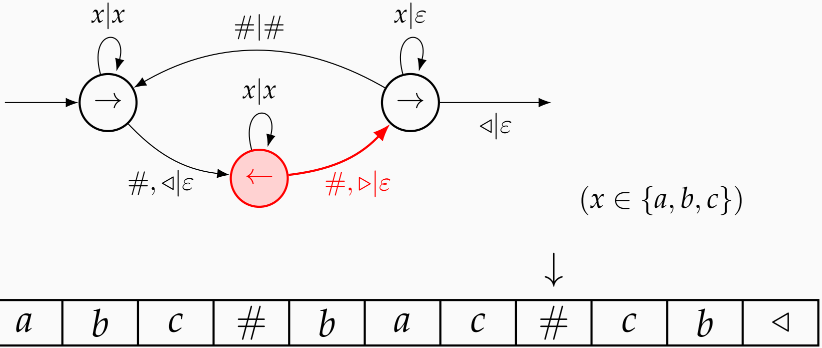
Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \mathtt{reverse}(w_1)\#\dots\#w_n \cdot \mathtt{reverse}(w_n)$



Output: $abccba\#baccab\#cbb$

The first definition of regular functions: (deterministic) two-way transducers

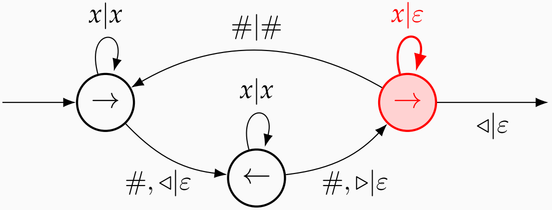
Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$



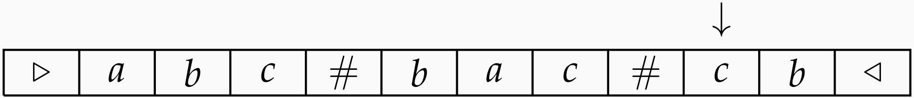
Output: $abccba\#baccab\#cbbc$

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$



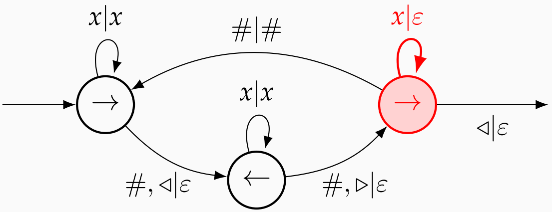
$(x \in \{a, b, c\})$



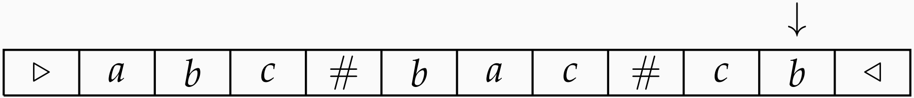
Output: $abccba\#baccab\#cbbc$

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



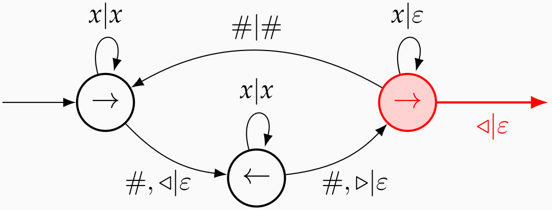
$(x \in \{a, b, c\})$



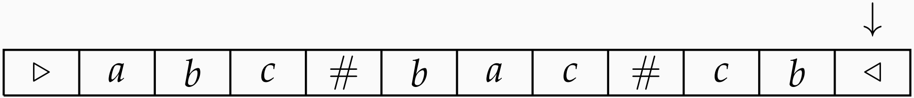
Output: $abccba\#baccab\#cbbc$

The first definition of regular functions: (deterministic) two-way transducers

Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \dots \# w_n \cdot \texttt{reverse}(w_n)$



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



Output: $abccba\#baccab\#cbbc$

Streaming string transducers = finite automata + string-valued registers

$$\begin{array}{rrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

a	c	a	b	$\#$	b	c	$\#$	c	a

$$X = \varepsilon \qquad Y = \varepsilon$$

Streaming string transducers = finite automata + string-valued registers

$$
\begin{aligned}
\texttt{mapReverse} : \quad \{a,b,c,\#\}^* &\quad\rightarrow\quad \{a,b,c,\#\}^* \\
w_1\#\ldots\#w_n &\quad\mapsto\quad \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n)
\end{aligned}
$$

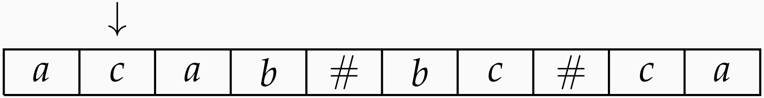
$\downarrow$

a	c	a	b	$\#$	b	c	$\#$	c	a

$$X = a \qquad Y = \varepsilon$$

Streaming string transducers = finite automata + string-valued registers

$$\begin{aligned}\texttt{mapReverse}: \quad \{a,b,c,\#\}^* &\to \{a,b,c,\#\}^* \\ w_1\#\ldots\#w_n &\mapsto \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n)\end{aligned}$$



$$X = ca \qquad Y = \varepsilon$$

Streaming string transducers = finite automata + string-valued registers

$$\begin{array}{rccc} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

$\downarrow$

a	c	a	b	$\#$	b	c	$\#$	c	a

$X = aca \qquad Y = \varepsilon$

Streaming string transducers = finite automata + string-valued registers

$$\texttt{mapReverse}: \{a,b,c,\#\}^* \to \{a,b,c,\#\}^*$$

$$w_1\#\ldots\#w_n \mapsto \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n)$$

$$\downarrow$$

a	c	a	b	#	b	c	#	c	a

$$X = baca \qquad Y = \varepsilon$$

Streaming string transducers = finite automata + string-valued registers

$$\begin{array}{rcl} \texttt{mapReverse}: \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

$$\downarrow$$

a	c	a	b	$\#$	b	c	$\#$	c	a

$$X = \varepsilon \qquad Y = baca\#$$

Streaming string transducers = finite automata + string-valued registers

$$\texttt{mapReverse}: \{a,b,c,\#\}^* \to \{a,b,c,\#\}^*$$
$$w_1\#\ldots\#w_n \mapsto \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n)$$

↓

a	c	a	b	$\#$	b	c	$\#$	c	a

$$X = b \qquad Y = baca\#$$

Streaming string transducers = finite automata + string-valued registers

$$\begin{aligned} \texttt{mapReverse}: \quad \{a,b,c,\#\}^* &\quad\to\quad \{a,b,c,\#\}^* \\ w_1\#\ldots\#w_n &\quad\mapsto\quad \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{aligned}$$

$\downarrow$

a	c	a	b	$\#$	b	c	$\#$	c	a

$$X = cb \qquad Y = baca\#$$

Streaming string transducers = finite automata + string-valued registers

$$\begin{aligned}\texttt{mapReverse}: \quad \{a,b,c,\#\}^* \quad &\to \quad \{a,b,c,\#\}^* \\ w_1\#\ldots\#w_n \quad &\mapsto \quad \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n)\end{aligned}$$

$\downarrow$

a	c	a	b	$\#$	b	c	$\#$	c	a

$$X = \varepsilon \qquad Y = baca\#cb\#$$

Streaming string transducers = finite automata + string-valued registers

$$\begin{aligned}\texttt{mapReverse}: \quad \{a,b,c,\#\}^* &\quad\to\quad \{a,b,c,\#\}^* \\ w_1\#\ldots\#w_n &\quad\mapsto\quad \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n)\end{aligned}$$

$\downarrow$

a	c	a	b	$\#$	b	c	$\#$	c	a

$X = c \qquad Y = baca\#cb\#$

$$\texttt{mapReverse} : \{a, b, c, \#\}^* \to \{a, b, c, \#\}^*$$
$$w_1\# \ldots \#w_n \mapsto \texttt{reverse}(w_1)\# \ldots \#\texttt{reverse}(w_n)$$

a	c	a	b	$\#$	b	c	$\#$	c	a

$$X = ac \qquad Y = baca\#cb\#$$

Streaming string transducers = finite automata + string-valued registers

$$\begin{aligned} \texttt{mapReverse}: \quad \{a,b,c,\#\}^* &\to \{a,b,c,\#\}^* \\ w_1\#\ldots\#w_n &\mapsto \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{aligned}$$

a	*c*	*a*	*b*	#	*b*	*c*	#	*c*	*a*

$X = ac \qquad Y = baca\#cb\# \qquad \texttt{mapReverse}(\ldots) = YX = baca\#cb\#ac$

Streaming string transducers = finite automata + string-valued registers

$$\begin{array}{rrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

a	c	a	b	$\#$	b	c	$\#$	c	a

$X = ac \qquad Y = baca\#cb\# \qquad \texttt{mapReverse}(\ldots) = YX = baca\#cb\#ac$

Regular functions = computed by copyless SSTs

$$a \mapsto \begin{cases} X := aX \\ Y := Y \end{cases} \qquad \# \mapsto \begin{cases} X := \varepsilon \\ Y := YX\# \end{cases}$$

each register appears *at most once* on the right of a := in a transition

Streaming string transducers = finite automata + string-valued registers

$$\begin{array}{rrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

a	c	a	b	$\#$	b	c	$\#$	c	a

$$X = ac \qquad Y = baca\#cb\# \qquad \texttt{mapReverse}(\ldots) = YX = baca\#cb\#ac$$

Regular functions = computed by copyless SSTs

$$a \mapsto \begin{cases} X := aX \\ Y := Y \end{cases} \qquad \# \mapsto \begin{cases} X := \varepsilon \\ Y := YX\# \end{cases}$$

each register appears *at most once* on the right of a := in a transition

$\rightsquigarrow$ connection with *linear logic* [Gallot, Lemay & Salvati 2020; N. & Pradic (in my PhD)]

Recognizing regular functions with functors on semigroups

A language is regular $\iff$ the corresponding decision problem factors as

$$\Sigma^* \xrightarrow{\text{some morphism}} \text{some finite semigroup} \to \{\text{yes}, \text{no}\}$$

The main theorem

A string-to-string function is regular $\iff$ it factors as

$$\Sigma^* \xrightarrow{\text{some morphism}} \mathsf{F}\Gamma^* \xrightarrow{\mathtt{out}_{\Gamma^*}} \Gamma^*$$

- for some *endofunctor* F on semigroups with *S finite $\Rightarrow F(S)$ finite*
- and some *natural transformation* $\mathtt{out} \colon \mathsf{UF} \Rightarrow \mathsf{U}$ (where U = forgetful to **Set**)

(Monoids instead of semigroups $\rightsquigarrow$ regular functions f such that $f(\varepsilon) = \varepsilon$)

Example

The following regular function maps *baa* to *cccaab*:

$$\{a,b\}^* \xrightarrow{\langle (\_ \mapsto c), \texttt{reverse} \rangle} \{a,b,c\}^* \times (\{a,b,c\}^*)^{\mathrm{op}} \xrightarrow{\text{concatenate}} \Sigma^*$$

- $S^{\mathrm{op}} = S$ where the product is reversed; $\texttt{reverse}\colon \Sigma^* \to (\Sigma^*)^{\mathrm{op}}$ is a morphism
- $\mathsf{F}S = S \times S^{\mathrm{op}}$ is a finiteness-preserving endofunctor
- $\cdot_S \colon S \times S^{\mathrm{op}} \to S$ is family of **Set**-functions natural in S

Some intuitions

A string-to-string function is regular $\iff$ it factors as

$$\Sigma^* \xrightarrow{\text{some morphism}} \mathsf{F}\Gamma^* \xrightarrow{\mathtt{out}_{\Gamma^*}} \Gamma^*$$

- for some *endofunctor* F on semigroups
- and some *natural transformation* $\mathtt{out} \colon \mathsf{UF} \Rightarrow \mathsf{U}$ (where U = forgetful to **Set**)
- with S finite $\Rightarrow F(S)$ finite

Some intuitions

A string-to-string function is regular $\iff$ it factors as

$$\Sigma^* \xrightarrow{\text{some morphism}} \mathsf{F}\Gamma^* \xrightarrow{\mathtt{out}_{\Gamma^*}} \Gamma^*$$

- for some *endofunctor* F on semigroups
  $\rightsquigarrow \mathsf{F}S =$ a data structure storing some elements from S, encoding compositional information
- and some *natural transformation* $\mathtt{out} \colon \mathsf{UF} \Rightarrow \mathsf{U}$ (where $\mathsf{U} =$ forgetful to **Set**)
- with S finite $\Rightarrow F(S)$ finite

Some intuitions

A string-to-string function is regular $\iff$ it factors as

$$\Sigma^* \xrightarrow{\text{some morphism}} \mathsf{F}\Gamma^* \xrightarrow{\mathtt{out}_{\Gamma^*}} \Gamma^*$$

- for some *endofunctor* F on semigroups
  $\rightsquigarrow$ $\mathsf{F}S$ = a data structure storing some elements from S, encoding compositional information
- and some *natural transformation* $\mathtt{out} \colon \mathsf{UF} \Rightarrow \mathsf{U}$ (where U = forgetful to **Set**)
  $\rightsquigarrow$ extract an element of S "uniformly" from $\mathsf{F}S$: procedure whose control flow should only depend on S-independent parts
- with S finite $\Rightarrow F(S)$ finite

Some intuitions

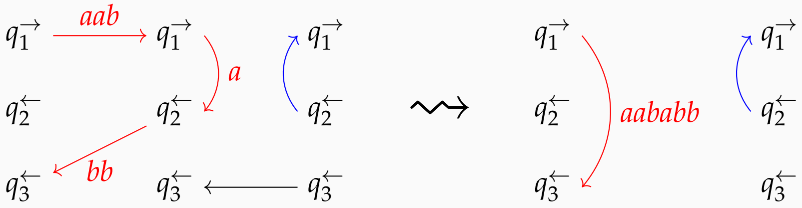
A string-to-string function is regular $\iff$ it factors as

$$\Sigma^* \xrightarrow{\text{some morphism}} \mathsf{F}\Gamma^* \xrightarrow{\mathtt{out}_{\Gamma^*}} \Gamma^*$$

- for some *endofunctor* F on semigroups
  $\rightsquigarrow$ $\mathsf{F}S$ = a data structure storing some elements from S, encoding compositional information
- and some *natural transformation* $\mathtt{out}\colon \mathsf{UF} \Rightarrow \mathsf{U}$ (where U = forgetful to **Set**)
  $\rightsquigarrow$ extract an element of S "uniformly" from $\mathsf{F}S$: procedure whose control flow should only depend on S-independent parts
- with S finite $\Rightarrow F(S)$ finite $\rightsquigarrow$ S-independent part $\simeq$ some *finite state*

Proof idea (1): two-way transducer $\longrightarrow$ functor

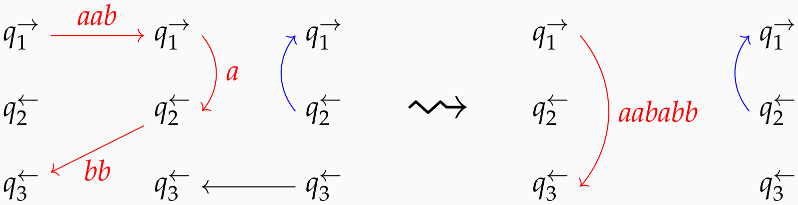
Behaviors of two-way transducers have a semigroup structure:



connection with traced monoidal categories: shapes = $\mathsf{Int}(\mathbf{Set}_{\text{partial}})(Q, Q)$ [Hines 2003]

Proof idea (1): two-way transducer $\longrightarrow$ functor

Behaviors of two-way transducers have a semigroup structure:



connection with traced monoidal categories: shapes = $\mathsf{Int}(\mathbf{Set}_{\text{partial}})(Q, Q)$ [Hines 2003]

Finitely many "shapes" $\rightsquigarrow$ finiteness-preserving $\mathsf{F}S = \sum_{\text{shapes}} S^{\text{number of labels}}$

(Actual proof in paper: similar phenomenon for streaming string transducers)

Proof idea (2): functor $\longrightarrow$ streaming string transducer

Key property of a "functorially recognized" function $f\colon \Sigma^* \to \Gamma^*$

For all $u, v \in \Sigma^*$, the parts of the output $f(uv)$ "caused by" the input prefix u consist of *a bounded number of factors* (contiguous subwords).

For $f\colon w \mapsto c^{|w|} \cdot \texttt{reverse}(w)$, at most 2 factors: $f(\underline{ba}a) = \underline{cc}ca\underline{ab}$

$\longrightarrow$ build a transducer whose registers store these factors after reading u

Proof idea (2): functor $\longrightarrow$ streaming string transducer

Key property of a "functorially recognized" function $f\colon \Sigma^* \to \Gamma^*$

For all $u, v \in \Sigma^*$, the parts of the output $f(uv)$ "caused by" the input prefix u consist of *a bounded number of factors* (contiguous subwords).

For $f\colon w \mapsto c^{|w|} \cdot \texttt{reverse}(w)$, at most 2 factors: $f(\underline{ba}a) = \underline{cc}ca\underline{ab}$

$\longrightarrow$ build a transducer whose registers store these factors after reading u

Formally: for f factored into $\Sigma^* \xrightarrow{h} \mathsf{F}\Gamma^* \xrightarrow{\mathtt{out}_{\Gamma^*}} \Gamma^*$, consider $(\oplus = \text{coproduct})$

$$\mathtt{out}\big(\mathsf{F}\underline{\iota}(h(ba)) \cdot \mathsf{F}\iota(h(a))\big) = \underline{cc} \cdot ca \cdot \underline{ab} \in \underline{\Sigma^*} \oplus \Sigma^*$$

Proof idea (2): functor $\longrightarrow$ streaming string transducer

Key property of a "functorially recognized" function $f\colon \Sigma^* \to \Gamma^*$

For all $u, v \in \Sigma^*$, the parts of the output $f(uv)$ "caused by" the input prefix u consist of *a bounded number of factors* (contiguous subwords).

For $f\colon w \mapsto c^{|w|} \cdot \texttt{reverse}(w)$, at most 2 factors: $f(\underline{ba}a) = \underline{cc}ca\underline{ab}$

$\longrightarrow$ build a transducer whose registers store these factors after reading u

Formally: for f factored into $\Sigma^* \xrightarrow{h} \mathsf{F}\Gamma^* \xrightarrow{\mathtt{out}_{\Gamma^*}} \Gamma^*$, consider (${\oplus} = $ coproduct)

$$\mathtt{out}\Big(\mathsf{F}\underline{\iota}(h(ba)) \cdot \mathsf{F}\iota(h(a))\Big) = \underline{cc} \cdot ca \cdot \underline{ab} \in \underline{\Sigma^*} \oplus \Sigma^*$$

Its "shape" $\underline{1} \cdot 1 \cdot \underline{1}$ is determined by $(\mathsf{F}\top(h(ba)), \mathsf{F}\top(h(a))) \in (\mathsf{F}1)^2$ $(\top : \Sigma^* \to 1)$

$+$ (1 finite $\implies$ $\mathsf{F}1$ finite) $\rightsquigarrow$ finitely many shapes $\rightsquigarrow$ desired bound

Conclusion

A language is regular $\iff$ the corresponding decision problem factors as

$$\Sigma^* \xrightarrow{\text{some morphism}} \text{some finite (monoid|semigroup)} \to \{\text{yes}, \text{no}\}$$

The main theorem

A string-to-string function is regular $\iff$ it factors as

$$\Sigma^* \xrightarrow{\text{some morphism}} \mathsf{F}\Gamma^* \xrightarrow{\mathtt{out}_{\Gamma^*}} \Gamma^*$$

- for some *endofunctor* F on semigroups with *S finite $\Rightarrow F(S)$ finite*
- and some *natural transformation* $\mathtt{out} \colon \mathsf{UF} \Rightarrow \mathsf{U}$ (where U = forgetful to **Set**)

Regular functions = computed by two-way transducers,
or copyless streaming string transducers, or...

Thanks for your attention!

A language is regular $\iff$ the corresponding decision problem factors as

$$\Sigma^* \xrightarrow{\text{some morphism}} \text{some finite (monoid|semigroup)} \to \{\text{yes}, \text{no}\}$$

The main theorem

A string-to-string function is regular $\iff$ it factors as

$$\Sigma^* \xrightarrow{\text{some morphism}} \mathsf{F}\Gamma^* \xrightarrow{\mathtt{out}_{\Gamma^*}} \Gamma^*$$

- for some *endofunctor* F on semigroups with *S finite $\Rightarrow F(S)$ finite*
- and some *natural transformation* $\mathtt{out} \colon \mathsf{UF} \Rightarrow \mathsf{U}$ (where U = forgetful to **Set**)

Regular functions = computed by two-way transducers,
or copyless streaming string transducers, or...