Algebraic Recognition of Regular Functions

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

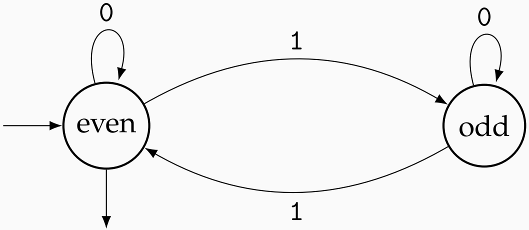
Séminaire Move, LIS, Université Aix-Marseille – 22 juin 2023

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 <u>star-free languages</u>

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 ($(\Leftarrow)$ 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

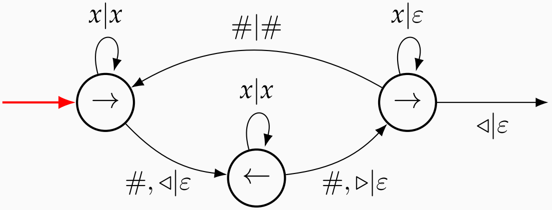
Regular functions are one of the most robust/canonical classes

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



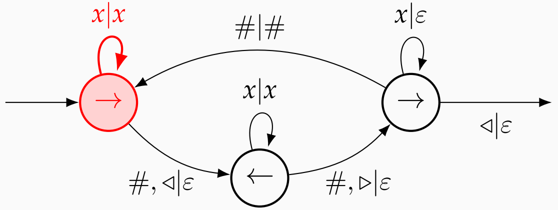
$(x \in \{a, b, c\})$

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

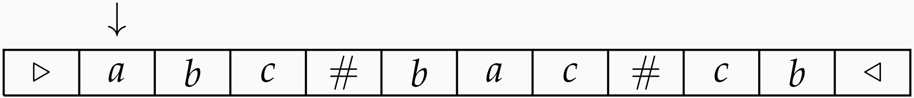
Output:

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

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



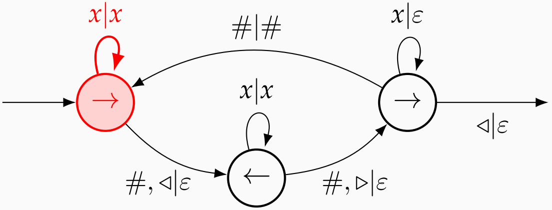
$(x \in \{a, b, c\})$



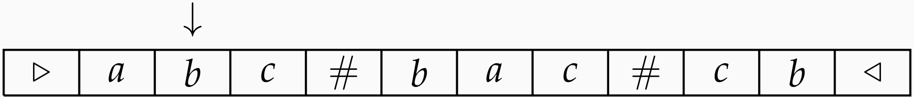
Output:

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

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



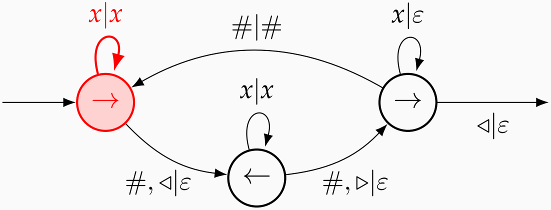
$(x \in \{a, b, c\})$



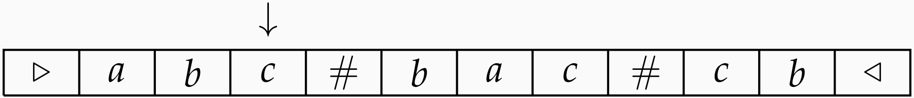
Output: a

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

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



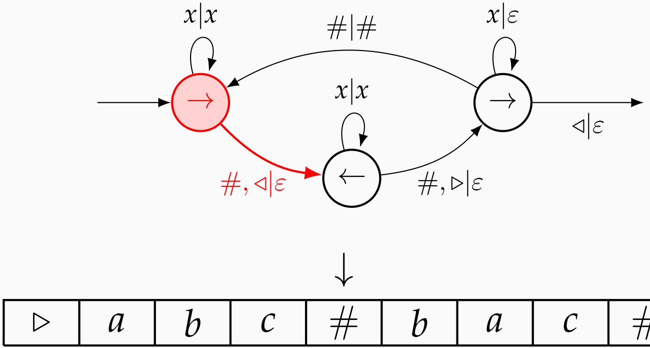
$(x \in \{a,b,c\})$



Output: ab

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

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

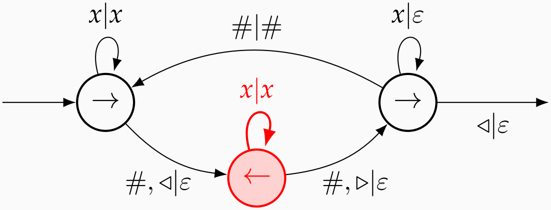


$(x \in \{a, b, c\})$

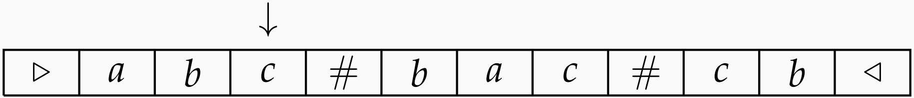
Output: abc

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

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



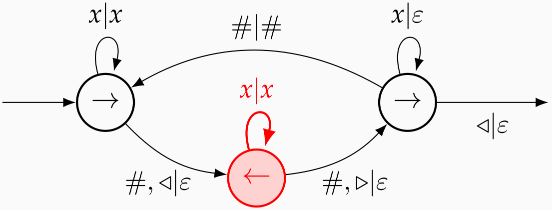
$(x \in \{a, b, c\})$



Output: abc

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

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



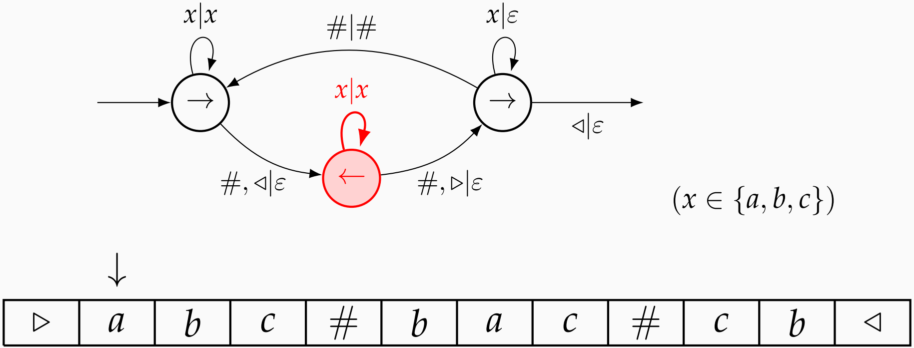
$(x \in \{a, b, c\})$

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

Output: $abcc$

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

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

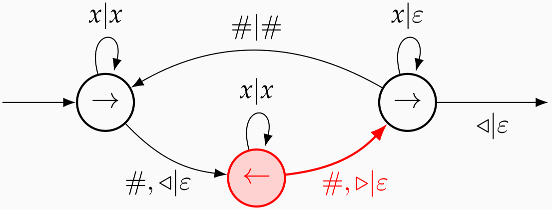


$(x \in \{a, b, c\})$

Output: $abccb$

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

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



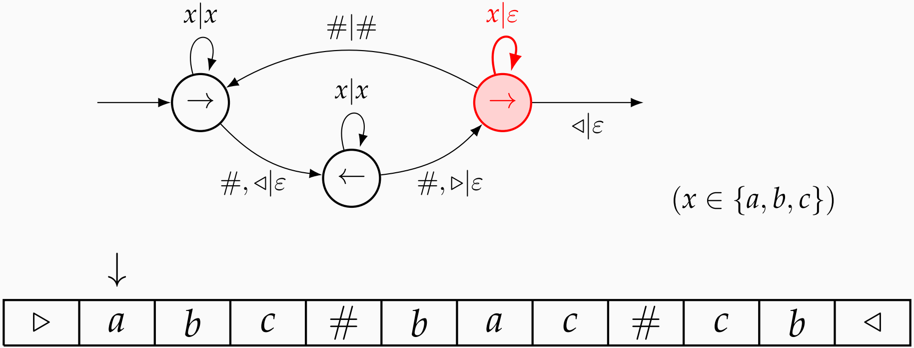
$(x \in \{a, b, c\})$

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

Output: *abccba*

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

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

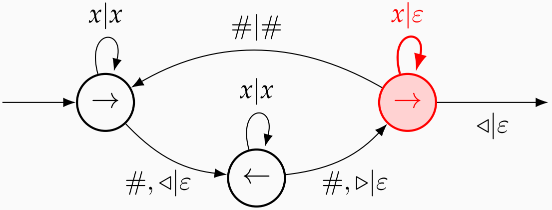


$(x \in \{a, b, c\})$

Output: *abccba*

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

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



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

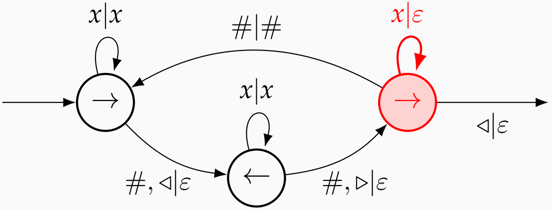
$\downarrow$

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

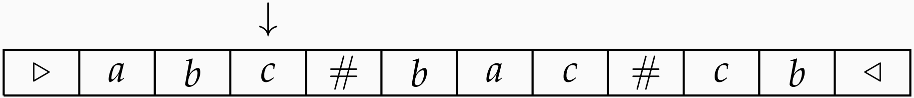
Output: *abccba*

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

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



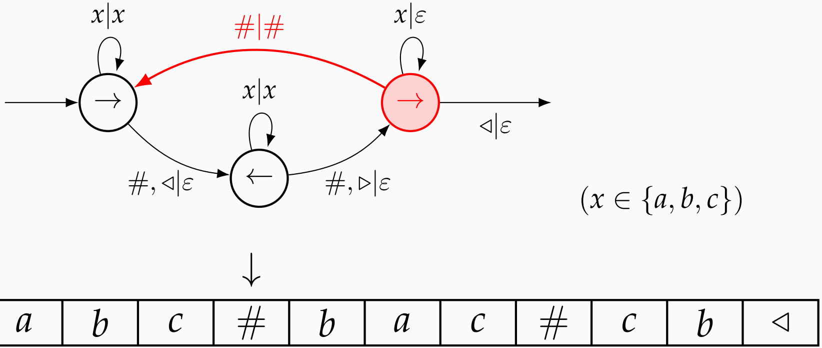
$(x \in \{a, b, c\})$



Output: *abccba*

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

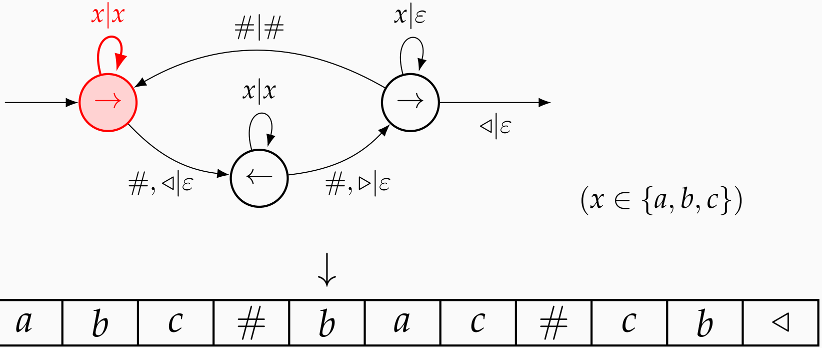
Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$



Output: $abccba$

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

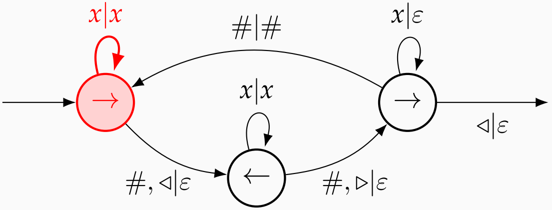
Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \mathtt{reverse}(w_1)\# \dots \# w_n \cdot \mathtt{reverse}(w_n)$



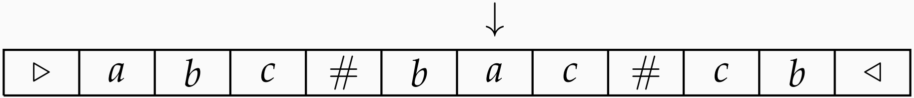
Output: $abccba\#$

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

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



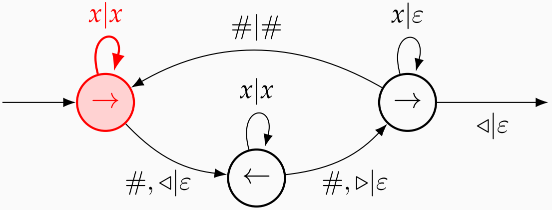
$(x \in \{a, b, c\})$



Output: $abccba\#b$

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

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



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

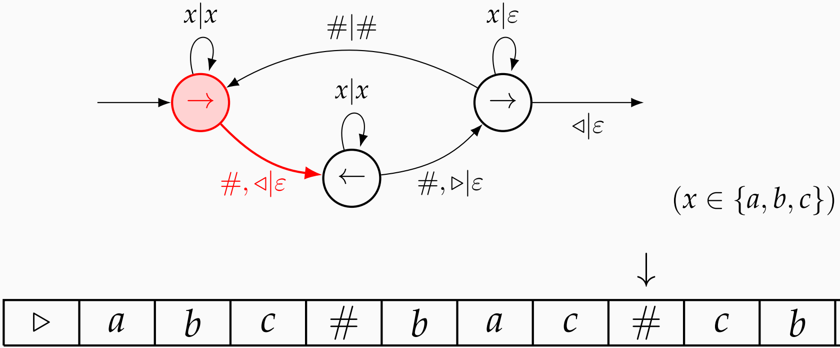
$\downarrow$

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

Output: $abccba\#ba$

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

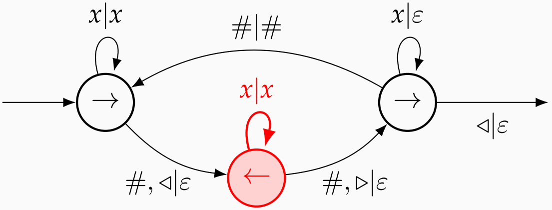
Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



Output: $abccba\#bac$

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

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



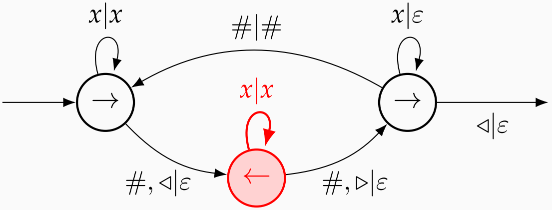
$(x \in \{a, b, c\})$

$\rhd$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\lhd$

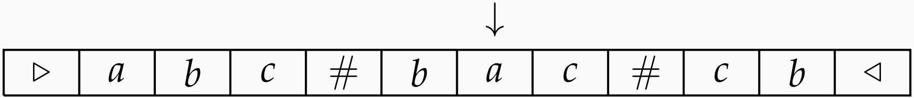
Output: $abccba\#bac$

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

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



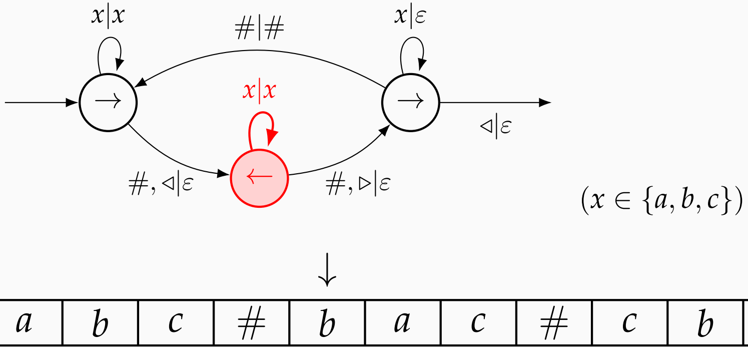
$(x \in \{a, b, c\})$



Output: $abccba\#bacc$

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

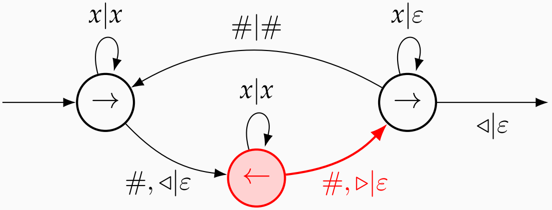
Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$



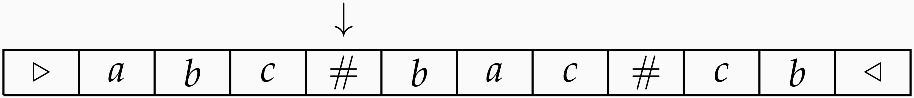
Output: $abccba\#bacca$

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

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



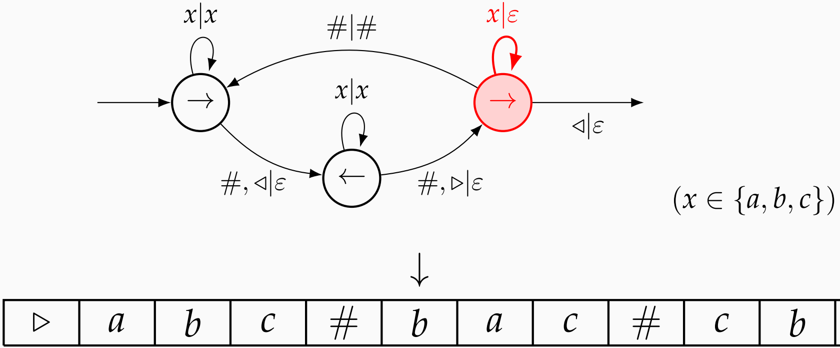
$(x \in \{a, b, c\})$



Output: $abccba\#baccab$

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

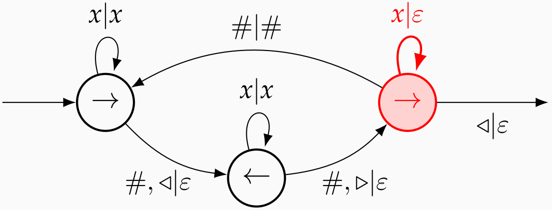
Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$



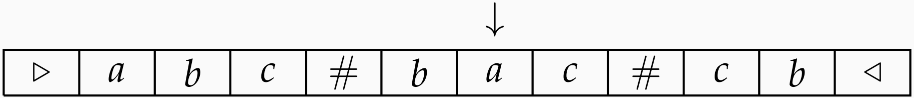
Output: $abccba\#baccab$

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

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



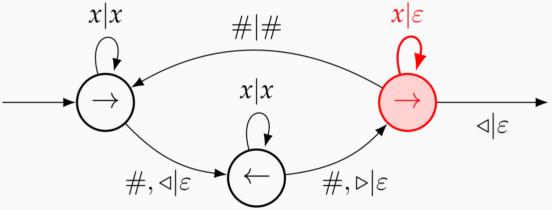
$(x \in \{a, b, c\})$



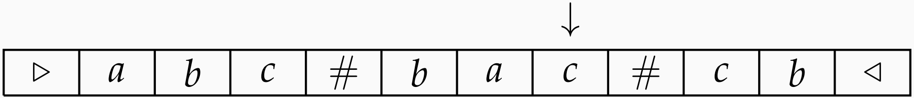
Output: $abccba\#baccab$

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

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



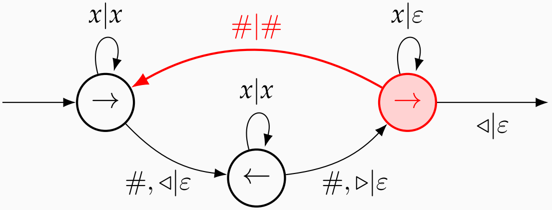
$(x \in \{a, b, c\})$



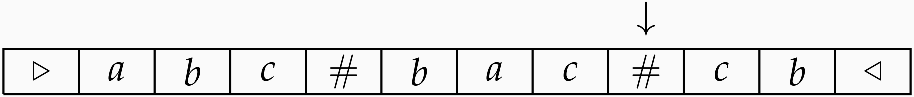
Output: $abccba\#baccab$

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

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



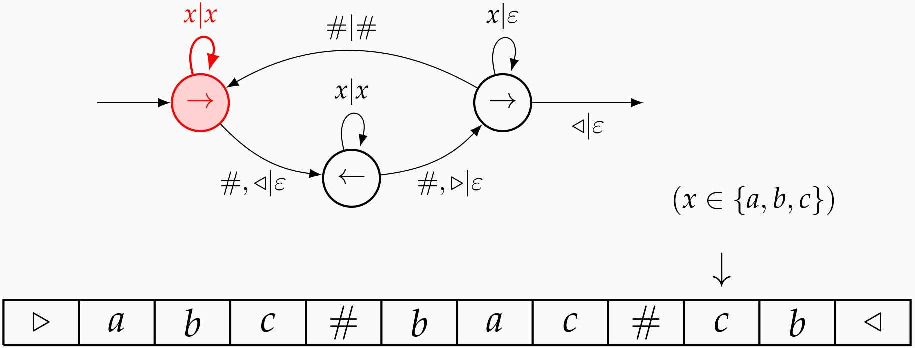
$(x \in \{a, b, c\})$



Output: $abccba\#baccab$

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

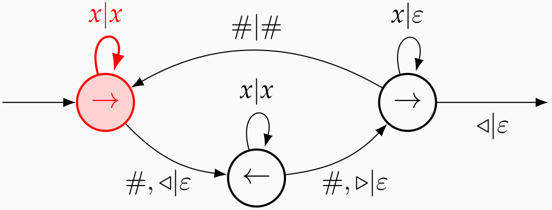
Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \mathtt{reverse}(w_1)\# \dots \# w_n \cdot \mathtt{reverse}(w_n)$



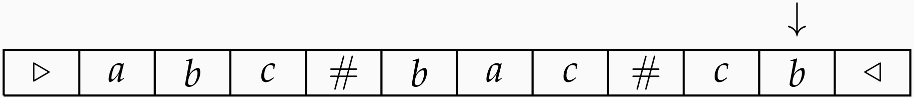
Output: $abccba\#baccab\#$

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

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



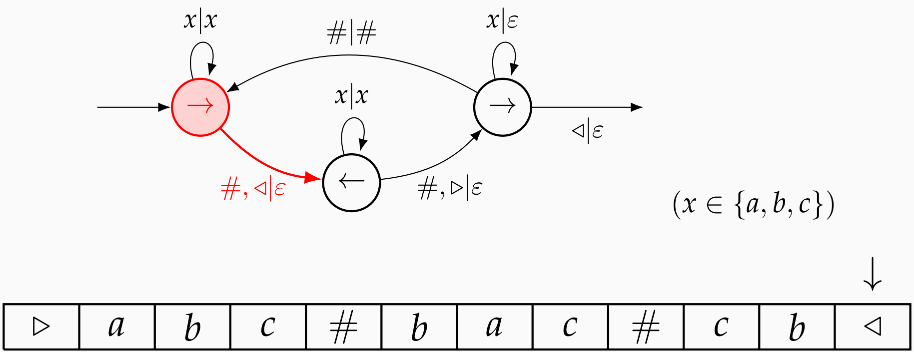
$(x \in \{a, b, c\})$



Output: $abccba\#baccab\#c$

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

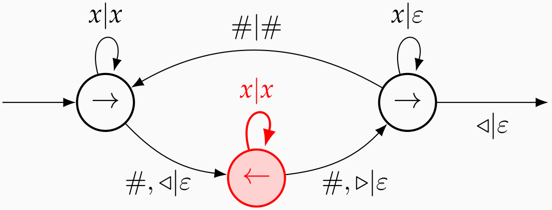
Example: $w_1\#\dots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\dots\#w_n \cdot \texttt{reverse}(w_n)$



Output: $abccba\#baccab\#cb$

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

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



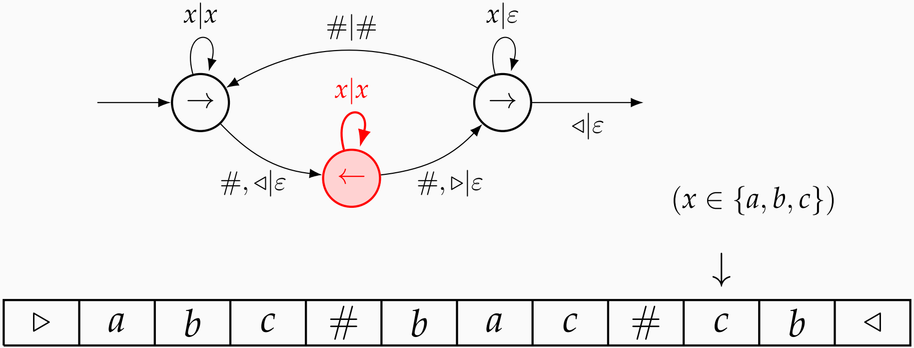
$(x \in \{a, b, c\})$

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

Output: $abccba\#baccab\#cb$

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

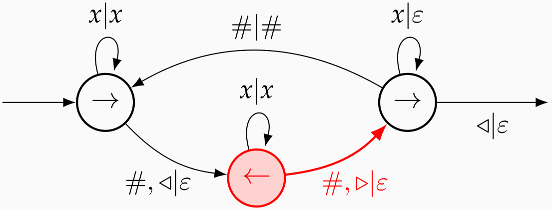
Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \dots \# w_n \cdot \texttt{reverse}(w_n)$



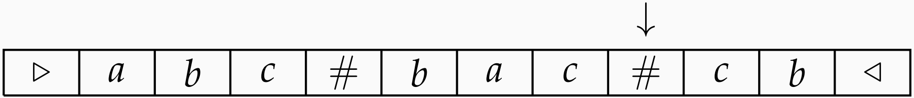
Output: $abccba\#baccab\#cbb$

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

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



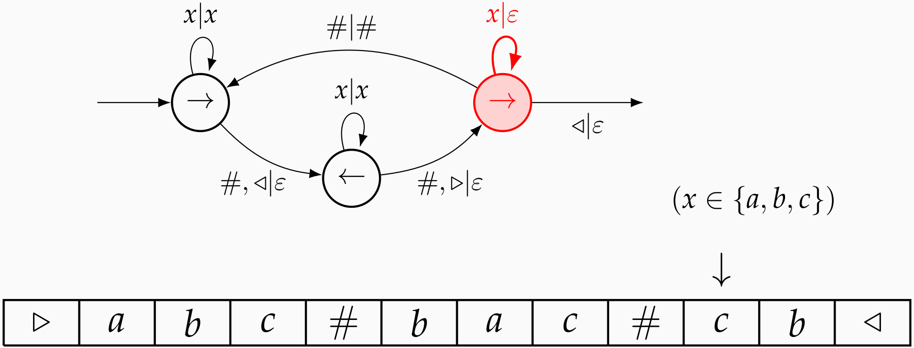
$(x \in \{a,b,c\})$



Output: $abccba\#baccab\#cbbc$

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

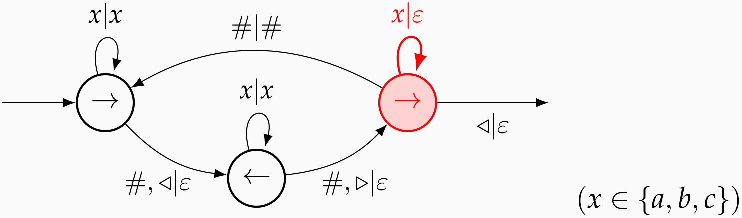
Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \mathtt{reverse}(w_1) \# \dots \# w_n \cdot \mathtt{reverse}(w_n)$



Output: $abccba\#baccab\#cbbc$

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

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

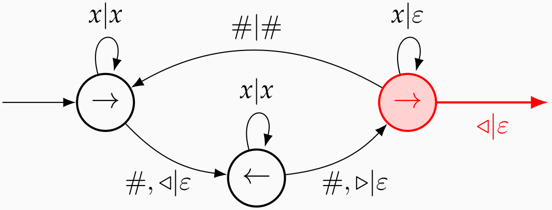


$\rhd$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\lhd$

Output: $abccba\#baccab\#cbbc$

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

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



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

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

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

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

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

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

Streaming string transducers = finite automata + string-valued registers

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

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

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

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

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

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

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

Streaming string transducers = finite automata + string-valued registers

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

$$\downarrow$$

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

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

Streaming string transducers = finite automata + string-valued registers

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

$\downarrow$

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

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

Streaming string transducers = finite automata + string-valued registers

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

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

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

Streaming string transducers = finite automata + string-valued registers

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

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

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

Streaming string transducers = finite automata + string-valued registers

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

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

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

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

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

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

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

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

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

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

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 idea

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 "construction on semigroups" F with S *finite* $\Rightarrow \mathsf{F}(S)$ *finite*
- and some "uniformly defined" $\mathtt{out}_A \colon \mathsf{F}(A) \to A$ (not a morphism)

Variants: concrete (registers, not in ICALP paper), short/abstract (category theory)

In both cases, easy to see *closure under composition*

Finitary register semigroups: example

$\mathsf{F}(S)$ has underlying set $\overbrace{\{0,1\}}^{\text{finite semigroup with } \times} \times \overbrace{S^{\{X,Y\}}}^{\text{contents of } S\text{-valued registers } X, Y}$; example in $\mathsf{F}(\mathbb{N}, +)$:

$$\left(1, \begin{pmatrix} X \mapsto 42 \\ Y \mapsto 218 \end{pmatrix}\right) \cdot \left(0, \begin{pmatrix} X \mapsto 1 \\ Y \mapsto 100 \end{pmatrix}\right) = \left(1 \times 0, \begin{pmatrix} X \mapsto 42 + 1 \\ Y \mapsto 42 + 100 \end{pmatrix}\right)$$

Finitary register semigroups: example

$\mathsf{F}(S)$ has underlying set $\overbrace{\{0,1\}}^{\text{finite semigroup with } \times} \times \overbrace{S^{\{X,Y\}}}^{\text{contents of } S\text{-valued registers } X,Y}$; example in $\mathsf{F}(\mathbb{N},+)$:

$$\left(1, \begin{pmatrix} X \mapsto 42 \\ Y \mapsto 218 \end{pmatrix}\right) \cdot \left(0, \begin{pmatrix} X \mapsto 1 \\ Y \mapsto 100 \end{pmatrix}\right) = \left(1 \times 0, \begin{pmatrix} X \mapsto 42+1 \\ Y \mapsto 42+100 \end{pmatrix}\right)$$

F defined from: *finite* "control" semigroup + registers + "associative" μ

$$\mu_{1,0}(X) = X_{\text{left}} X_{\text{right}} \qquad \mu_{1,0}(Y) = X_{\text{left}} Y_{\text{right}} \qquad \ldots$$

Exercise

Using this F, complete μ and find a homomorphism h so that

$$f \colon \{a,b,c\}^* \xrightarrow{h} \mathsf{F}(\{a,b\}^*) \xrightarrow{\text{value of register } Y} \{a,b\}^*$$

satisfies $\forall u \in \{a,b,c\}^*,\ \forall v \in \{a,b\}^*,\ f(ucv) = a^{|u|}bv$ and $f(v) = v$.

From streaming string transducers to finitary register semigroups

Decomposition of register updates:

$$\begin{cases} X := abXcY \\ Y := ba \end{cases} \quad \rightsquigarrow \quad \text{shape} \begin{cases} X := Z_1 X Z_2 Y \\ Y := Z_3 \end{cases} \quad + \quad \text{labels } Z_1 = ab, \ldots$$

copyless SST $\implies$ *bounded-copy* SST $\iff$ *finitely many* possible shapes

Theorem

Bounded-copy streaming string transducers = regular functions

From streaming string transducers to finitary register semigroups

Decomposition of register updates:

$$\begin{cases} X := abXcY \\ Y := ba \end{cases} \quad \rightsquigarrow \quad \text{shape} \begin{cases} X := Z_1 X Z_2 Y \\ Y := Z_3 \end{cases} \quad + \quad \text{labels } Z_1 = ab, \dots$$

copyless SST $\Longrightarrow$ *bounded-copy* SST $\Longleftrightarrow$ *finitely many* possible shapes

Theorem

Bounded-copy streaming string transducers = regular functions

$\mathsf{F}(S)$ = semigroup of state+register updates with "coefficients" in S

$\rightsquigarrow$ represent as register semigroup whose underlying finite sg uses shapes

From streaming string transducers to finitary register semigroups

Decomposition of register updates:

$$\begin{cases} X := abXcY \\ Y := ba \end{cases} \quad \rightsquigarrow \quad \text{shape} \begin{cases} X := Z_1 X Z_2 Y \\ Y := Z_3 \end{cases} \quad + \quad \text{labels } Z_1 = ab, \ldots$$

$$\textit{copyless}\ \text{SST} \implies \textit{bounded-copy}\ \text{SST} \iff \textit{finitely many}\ \text{possible shapes}$$

Theorem

Bounded-copy streaming string transducers = regular functions

$\mathsf{F}(S)$ = semigroup of state+register updates with "coefficients" in S

$\rightsquigarrow$ represent as register semigroup whose underlying finite sg uses shapes

Remark: $\exists$ translation: *reversible* two-way transd. $\longrightarrow$ "copyless" register sg

(via "two-sided Shepherdson construction")

From finitary register semigroups to bounded-copy SST

Register semigroup $(S_\mathsf{F}, R_\mathsf{F}, \mu_\mathsf{F})$ + morphism $h\colon \Sigma^* \to \mathsf{F}(\Gamma^*) \rightsquigarrow$ "naive" SST

- set of states S_F, registers R_F — therefore, configurations $\approx \mathsf{F}(\Gamma^*)$
- transition for $c \in \Sigma \approx$ action of $h(c)$ (as in finite monoid $\to$ DFA translation)
  $\implies$ after reading an input prefix w, current configuration $\approx h(w)$

From finitary register semigroups to bounded-copy SST

Register semigroup $(S_{\mathsf{F}}, R_{\mathsf{F}}, \mu_{\mathsf{F}})$ + morphism $h\colon \Sigma^* \to \mathsf{F}(\Gamma^*) \rightsquigarrow$ “naive” SST

- set of states S_{F}, registers R_{F} — therefore, configurations $\approx \mathsf{F}(\Gamma^*)$
- transition for $c \in \Sigma \approx$ action of $h(c)$ (as in finite monoid $\to$ DFA translation)

$\implies$ after reading an input prefix w, current configuration $\approx h(w)$

Key property

This streaming string transducer is *automatically* bounded-copy (because the register update “shape” of $h(w)$ is determined by S_{F} part).

[propaganda time]

Abstracting further using categories

A *category* = some *objects* with *arrows* between them

+ can take composition $g \circ f$ when $\mathrm{source}(g) = \mathrm{target}(f)$ + identity arrows

Examples

$\underbrace{\text{Sets and functions}}_{\text{“the category of sets”}}$ / sets and relations / $\underbrace{\text{semigroups and homomorphisms}}_{\text{“the category of semigroups”}}$ / ...

Abstracting further using categories

A *category* = some *objects* with *arrows* between them

+ can take composition $g \circ f$ when $\mathrm{source}(g) = \mathrm{target}(f)$ + identity arrows

Examples

$\underbrace{\text{Sets and functions}}_{\text{“the category of sets”}}$ / sets and relations / $\underbrace{\text{semigroups and homomorphisms}}_{\text{“the category of semigroups”}}$ / ...

Functors = “morphisms between categories”

F maps objects to objects, arrows $f\colon A \to B$ to $\mathsf{F}(f)\colon \mathsf{F}(A) \to \mathsf{F}(B)$, preserves $\circ$/id

- semigroup-to-set *forgetful* functor: A semigroup $\mapsto$ underlying set of A
- set-to-semigroup $A \mapsto A^*$
- semigroup-to-semigroup $A \mapsto A^2$ or $A \mapsto A^{\mathrm{op}}$ or ...
- etc.

Natural transformations

Let $F, G \colon \mathcal{C} \to \mathcal{D}$ be functors. A family of arrows $\eta_A \colon \mathsf{F}(A) \to \mathsf{G}(A)$ is *natural* when

$$\forall f \colon A \to B,\ \eta_B \circ \mathsf{F}(f) = \mathsf{G}(f) \circ \eta_A$$

Typical example: generic functions between data structures

$\mathsf{List}(A) = A^*$, $\mathsf{List}(f)([a_1, \ldots, a_n]) = [f(a_1), \ldots, f(a_n)]$ $\mathsf{Maybe}(A) = \{\mathtt{None}\} + A$, ...

$$\eta_A \colon x \in \mathsf{Maybe}(A) \mapsto [x, x] \text{ if } x \in A \text{ else } [] \in \mathsf{List}(A)$$

$$\begin{tikzcd}
a \arrow[r, "\eta_A"] \arrow[d, "\mathsf{Maybe}(f)"'] & {[a, a]} \arrow[d, "\mathsf{List}(f)"] \\
f(a) \arrow[r, "\eta_B"] & {[f(a), f(a)]}
\end{tikzcd} \qquad (a \neq \mathtt{None})$$

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 – category-theoretic version

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

($\Rightarrow$) doable using finitary register semigroups

Non-trivial proof of ($\Leftarrow$) morally extracting the "origin semantics" of the function

 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 – category-theoretic version

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

($\Rightarrow$) doable using finitary register semigroups

Non-trivial proof of ($\Leftarrow$) morally extracting the “origin semantics” of the function

Proof idea: 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: 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 $f\colon w \mapsto c^{|w|} \cdot \mathtt{reverse}(w)$, 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)

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

Proof idea: 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 $(\oplus = \text{coproduct})$

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

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