Geometry of Interaction meets actually existing automata

Lê Thành Dũng (Tito) Nguyễn — `nltd@nguyentito.eu` — ÉNS Lyon

Lambda Pros day @ IRIF — June 28th, 2023

Overview

> *The GoI* [Geometry of Interaction] *view of computation makes it possible to interpret computation as a token machine that traverses a graph strongly related to the syntactic structure of the term. Somewhat suprisingly, so far this nearly automata-theoretic flavour of GoI has not been exploited to establish connections with automata models* [...]
>
> — Clairambault & Murawski, MFCS 2019

Overview

> *The GoI* [Geometry of Interaction] *view of computation makes it possible to interpret computation as a token machine that traverses a graph strongly related to the syntactic structure of the term. Somewhat suprisingly, so far this nearly automata-theoretic flavour of GoI has not been exploited to establish connections with automata models* [...]
>
> — Clairambault & Murawski, MFCS 2019

- [Hines 2003, Katsumata 2008]: connections between *categorical* GoI & automata
  inspired a new automaton model... which **Cécilia Pradic** and I studied

Overview

> *The GoI* [Geometry of Interaction] *view of computation makes it possible to interpret computation as a token machine that traverses a graph strongly related to the syntactic structure of the term. Somewhat suprisingly, so far this nearly automata-theoretic flavour of GoI has not been exploited to establish connections with automata models* […]
>
> — Clairambault & Murawski, MFCS 2019

- [Hines 2003, Katsumata 2008]: connections between *categorical* GoI & automata inspired a new automaton model… which **Cécilia Pradic** and I studied
- Applications of token-machine GoI to "implicit automata" (cf. Cécilia's talk), or "higher-order transducers" (à la [Gallot, Lemay & Salvati 2020]): ongoing work with **Gabriele Vanoni**

Reminder: two-way automata (1)

Transitions: update finite state + move left/right depending on new state

Example: states $Q = \{q_1^{\rightarrow}, q_2^{\leftarrow}, q_3^{\leftarrow}\}$, initial state $q_1^{\rightarrow}$

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$

Directed states are an old idea,[1] needed to define *reversible* two-way automata

---

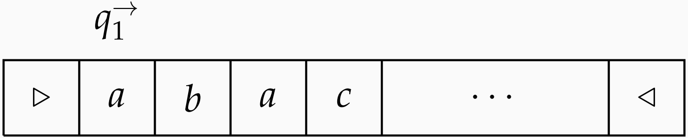
[1] cf. e.g. J.-C. Birget, *Concatenation of Inputs in a Two-Way Automaton* (1989)

Reminder: two-way automata (1)

Transitions: update finite state + move left/right depending on new state

Example: states $Q = \{q_1^{\rightarrow}, q_2^{\leftarrow}, q_3^{\leftarrow}\}$, initial state $q_1^{\rightarrow}$

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$



Directed states are an old idea,[1] needed to define *reversible* two-way automata

---

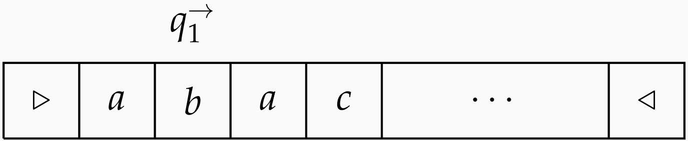
[1] cf. e.g. J.-C. Birget, *Concatenation of Inputs in a Two-Way Automaton* (1989)

Reminder: two-way automata (1)

Transitions: update finite state + move left/right depending on new state

Example: states $Q = \{q_1^{\rightarrow}, q_2^{\leftarrow}, q_3^{\leftarrow}\}$, initial state $q_1^{\rightarrow}$

$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow}$ $\qquad$ $q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow}$ $\qquad$ $q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow}$ $\qquad$ $q_3^{\leftarrow}, b \mapsto \text{accept}$



Directed states are an old idea,[1] needed to define *reversible* two-way automata

---

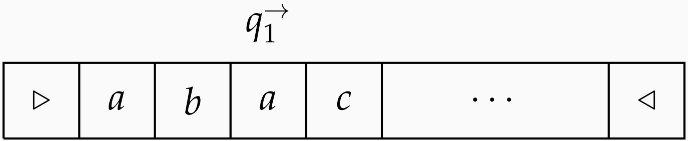
[1] cf. e.g. J.-C. Birget, *Concatenation of Inputs in a Two-Way Automaton* (1989)

Reminder: two-way automata (1)

Transitions: update finite state + move left/right depending on new state

Example: states $Q = \{q_1^{\rightarrow}, q_2^{\leftarrow}, q_3^{\leftarrow}\}$, initial state $q_1^{\rightarrow}$

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$


$q_1^{\rightarrow}$

$\triangleright$	a	b	a	c	$\cdots$	$\triangleleft$

Directed states are an old idea,[1] needed to define *reversible* two-way automata

---

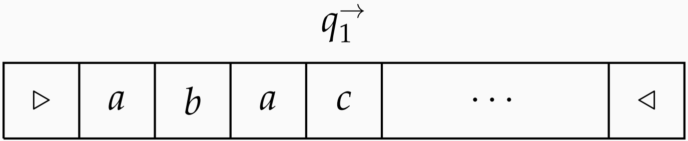
[1] cf. e.g. J.-C. Birget, *Concatenation of Inputs in a Two-Way Automaton* (1989)

Reminder: two-way automata (1)

Transitions: update finite state + move left/right depending on new state

Example: states $Q = \{q_1^{\rightarrow}, q_2^{\leftarrow}, q_3^{\leftarrow}\}$, initial state $q_1^{\rightarrow}$

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$



Directed states are an old idea,[1] needed to define *reversible* two-way automata

---

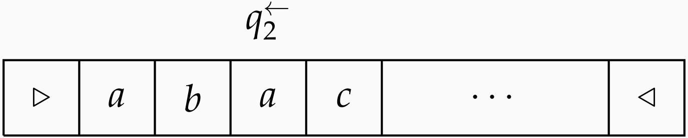
[1] cf. e.g. J.-C. Birget, *Concatenation of Inputs in a Two-Way Automaton* (1989)

Reminder: two-way automata (1)

Transitions: update finite state + move left/right depending on new state

Example: states $Q = \{q_1^{\rightarrow}, q_2^{\leftarrow}, q_3^{\leftarrow}\}$, initial state $q_1^{\rightarrow}$

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$



Directed states are an old idea,[1] needed to define *reversible* two-way automata

---

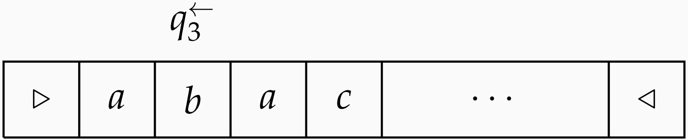
[1] cf. e.g. J.-C. Birget, *Concatenation of Inputs in a Two-Way Automaton* (1989)

Reminder: two-way automata (1)

Transitions: update finite state + move left/right depending on new state

Example: states $Q = \{q_1^{\rightarrow}, q_2^{\leftarrow}, q_3^{\leftarrow}\}$, initial state $q_1^{\rightarrow}$

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$



Directed states are an old idea,[1] needed to define *reversible* two-way automata

---

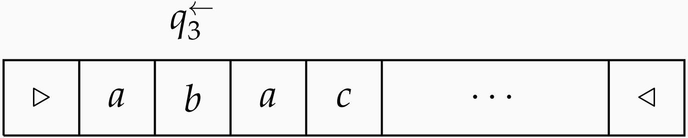
[1] cf. e.g. J.-C. Birget, *Concatenation of Inputs in a Two-Way Automaton* (1989)

Reminder: two-way automata (1)

Transitions: update finite state + move left/right depending on new state

Example: states $Q = \{q_1^{\rightarrow}, q_2^{\leftarrow}, q_3^{\leftarrow}\}$, initial state $q_1^{\rightarrow}$

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$



Directed states are an old idea,[1] needed to define *reversible* two-way automata

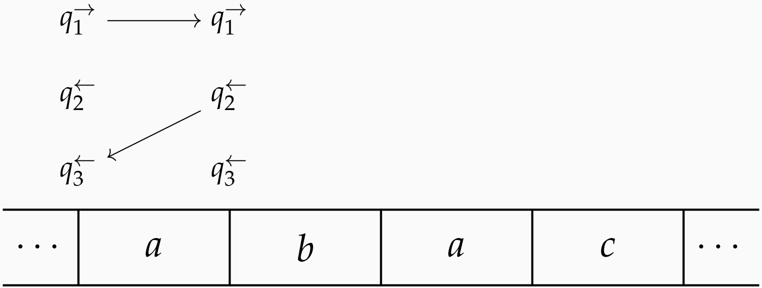
---

[1] cf. e.g. J.-C. Birget, *Concatenation of Inputs in a Two-Way Automaton* (1989)

Reminder: two-way automata (2)

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$

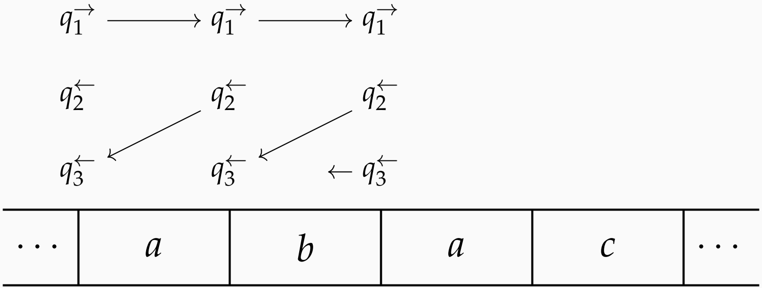
Graphical representation of transitions for each letter:



Reminder: two-way automata (2)

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$

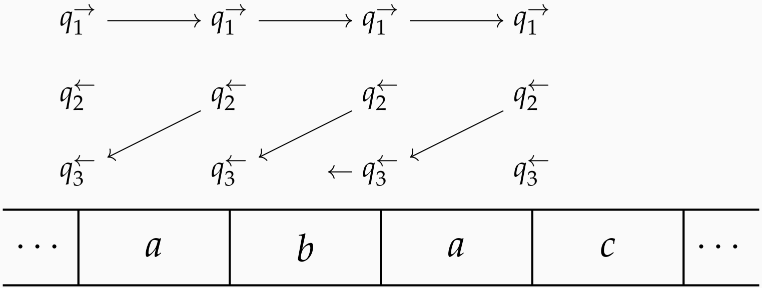
Graphical representation of transitions for each letter:



Reminder: two-way automata (2)

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$

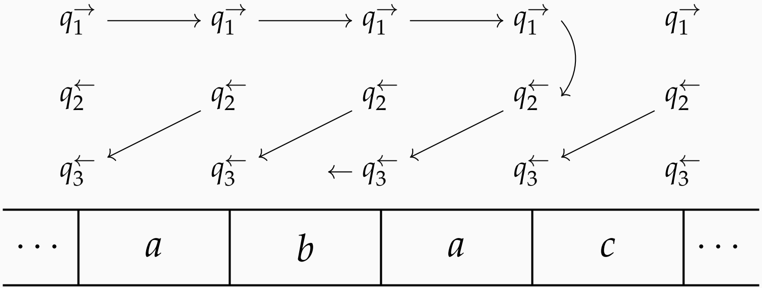
Graphical representation of transitions for each letter:



Reminder: two-way automata (2)

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$

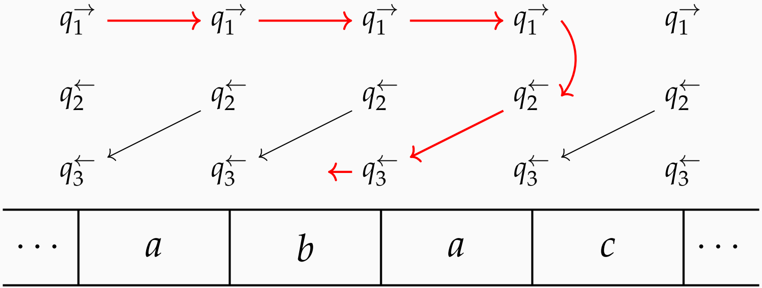
Graphical representation of transitions for each letter:



Reminder: two-way automata (2)

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$

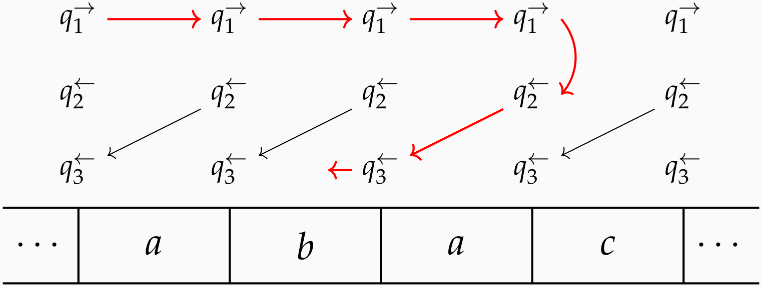
Graphical representation of transitions for each letter:



Reminder: two-way automata (2)

$$q_1^{\rightarrow}, (a|b) \mapsto q_1^{\rightarrow} \qquad q_1^{\rightarrow}, c \mapsto q_2^{\leftarrow} \qquad q_2^{\leftarrow}, (a|b|c) \mapsto q_3^{\leftarrow} \qquad q_3^{\leftarrow}, b \mapsto \text{accept}$$

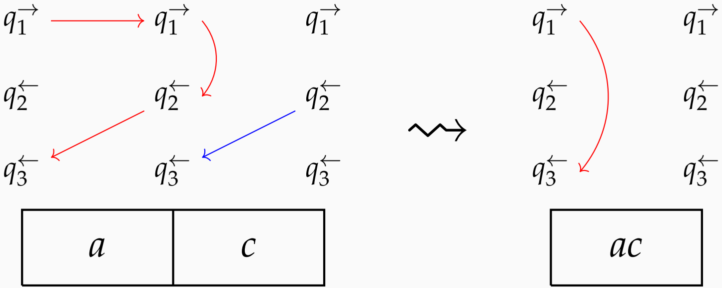
Graphical representation of transitions for each letter:



This is *deterministic*: outdegree ≤ 1; even *reversible*: deterministic + indegree ≤ 1

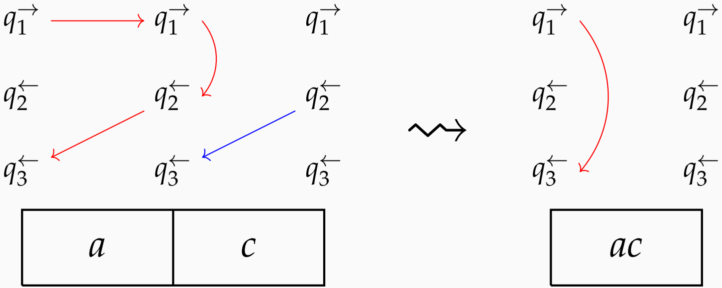
Reminder: two-way automata (3)

Behaviors (or crossing types) form a monoid:



Reminder: two-way automata (3)

Behaviors (or crossing types) form a monoid:



This monoid is finite, therefore two-way automata recognize regular languages (modern account of "two-sided" variant of Shepherdson's construction (1959))

Reversible behaviors: closed under product, can recognize all regular languages

Describing the monoid of behaviors

Let $Q = Q^{\leftarrow} \cup Q^{\rightarrow}$ be the set of directed states.

- A behavior is given by 4 partial functions

$$f_{\text{left}\to\text{right}} \colon Q^{\rightarrow} \rightharpoonup Q^{\rightarrow} \qquad f_{\text{left}\to\text{left}} \colon Q^{\rightarrow} \rightharpoonup Q^{\leftarrow} \qquad f_{\text{right}\to\text{left}} \colon Q^{\leftarrow} \rightharpoonup Q^{\leftarrow} \qquad \ldots$$

- Composition is a complicated thing that “follows paths”

Describing the monoid of behaviors

Let $Q = Q^{\leftarrow} \cup Q^{\rightarrow}$ be the set of directed states.

- A behavior is given by 4 partial functions

$$f_{\text{left}\to\text{right}} \colon Q^{\rightarrow} \rightharpoonup Q^{\rightarrow} \qquad f_{\text{left}\to\text{left}} \colon Q^{\rightarrow} \rightharpoonup Q^{\leftarrow} \qquad f_{\text{right}\to\text{left}} \colon Q^{\leftarrow} \rightharpoonup Q^{\leftarrow} \qquad \ldots$$

- Composition is a complicated thing that "follows paths"

This is the case $A = B = Q$ of $A^{\rightarrow} + B^{\leftarrow} \rightharpoonup A^{\leftarrow} + B^{\rightarrow}$ since it's equivalent to

$$f_{\text{left}\to\text{right}} \colon A^{\rightarrow} \rightharpoonup B^{\rightarrow} \qquad f_{\text{left}\to\text{left}} \colon A^{\rightarrow} \rightharpoonup A^{\leftarrow} \qquad f_{\text{right}\to\text{left}} \colon B^{\leftarrow} \rightharpoonup A^{\leftarrow} \qquad \ldots$$

Describing the monoid of behaviors

Let $Q = Q^{\leftarrow} \cup Q^{\rightarrow}$ be the set of directed states.

- A behavior is given by 4 partial functions

$$f_{\text{left}\to\text{right}} \colon Q^{\rightarrow} \rightharpoonup Q^{\rightarrow} \qquad f_{\text{left}\to\text{left}} \colon Q^{\rightarrow} \rightharpoonup Q^{\leftarrow} \qquad f_{\text{right}\to\text{left}} \colon Q^{\leftarrow} \rightharpoonup Q^{\leftarrow} \qquad \ldots$$

- Composition is a complicated thing that "follows paths"

This is the case $A = B = Q$ of $A^{\rightarrow} + B^{\leftarrow} \rightharpoonup A^{\leftarrow} + B^{\rightarrow}$ since it's equivalent to

$$f_{\text{left}\to\text{right}} \colon A^{\rightarrow} \rightharpoonup B^{\rightarrow} \qquad f_{\text{left}\to\text{left}} \colon A^{\rightarrow} \rightharpoonup A^{\leftarrow} \qquad f_{\text{right}\to\text{left}} \colon B^{\leftarrow} \rightharpoonup A^{\leftarrow} \qquad \ldots$$

Abstract even further: $\mathsf{PSet}(A^{\rightarrow} + B^{\leftarrow}, A^{\leftarrow} + B^{\rightarrow})$ (cat. of partial fn.)

for reversible behaviours, replace with PInj (partial injections)

The general category-theoretic construction

Let $(\mathcal{C}, \otimes)$ be a monoidal category, e.g. $(\mathsf{PSet}, +)$. Consider a new category:

- an object: $A = (A^+, A^-)$ for $A^+, A^- \in \mathcal{C}$
- a morphism $A \to B$ consists of a $\mathcal{C}$-morphism $A^+ \otimes B^- \to A^- \otimes B^+$

The general category-theoretic construction

Let $(\mathcal{C}, \otimes)$ be a monoidal category, e.g. $(\mathsf{PSet}, +)$. Consider a new category:

- an object: $A = (A^+, A^-)$ for $A^+, A^- \in \mathcal{C}$
- a morphism $A \to B$ consists of a $\mathcal{C}$-morphism $A^+ \otimes B^- \to A^- \otimes B^+$
- composition... is complicated, and requires more structure on $\mathcal{C}$

The general category-theoretic construction

Let $(\mathcal{C}, \otimes)$ be a monoidal category, e.g. $(\mathsf{PSet}, +)$. Consider a new category:

- an object: $A = (A^+, A^-)$ for $A^+, A^- \in \mathcal{C}$
- a morphism $A \to B$ consists of a $\mathcal{C}$-morphism $A^+ \otimes B^- \to A^- \otimes B^+$
- composition… is complicated, and requires more structure on $\mathcal{C}$

$\rightsquigarrow$ Int-construction on traced monoidal categories! [Joyal, Street & Verity 1996]

- $\mathsf{Int}(\mathcal{C})$ is automatically a semantics of linear λ-calculus (compact closed category)
- as observed by Abramsky, $\mathsf{Int}(\mathsf{PInj}) \simeq$ Girard's original GoI semantics

The general category-theoretic construction

Let $(\mathcal{C}, \otimes)$ be a monoidal category, e.g. $(\mathsf{PSet}, +)$. Consider a new category:

- an object: $A = (A^+, A^-)$ for $A^+, A^- \in \mathcal{C}$
- a morphism $A \to B$ consists of a $\mathcal{C}$-morphism $A^+ \otimes B^- \to A^- \otimes B^+$
- composition… is complicated, and requires more structure on $\mathcal{C}$

$\rightsquigarrow$ Int-construction on traced monoidal categories! [Joyal, Street & Verity 1996]

- $\mathsf{Int}(\mathcal{C})$ is automatically a semantics of linear λ-calculus (compact closed category)
- as observed by Abramsky, $\mathsf{Int}(\mathsf{PInj}) \simeq$ Girard's original GoI semantics

$(\mathsf{Int}(\mathsf{PSet})|\mathsf{Int}(\mathsf{PInj}))$-automata = (deterministic|reversible) two-way automata

[Hines 2003], rephrased using the $\mathcal{C}$-automata of [Colcombet & Petrişan 2017]

Planar geometry of interaction

Just like our two-way behaviours, morphisms in $\mathsf{Int}(\mathsf{PSet})$ / $\mathsf{Int}(\mathsf{PInj})$ can be drawn as diagrams. Categories of *planar* diagrams ↔ *non-commutative* linear λ-calculus

- order of arguments matters: $\lambda x.\,\lambda y.\,t$ "must use x before y"
- equivalently, syntax tree with binding edges is planar…

Planar geometry of interaction

Just like our two-way behaviours, morphisms in $\mathsf{Int}(\mathsf{PSet})$ / $\mathsf{Int}(\mathsf{PInj})$ can be drawn as diagrams. Categories of *planar* diagrams $\leftrightarrow$ *non-commutative* linear λ-calculus

- order of arguments matters: $\lambda x.\, \lambda y.\, t$ “must use x before y”
- equivalently, syntax tree with binding edges is planar…

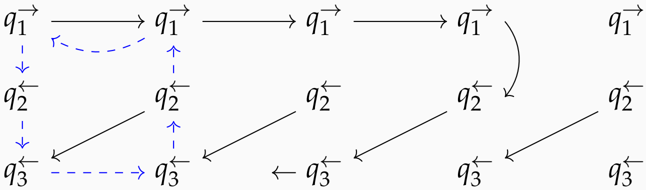
[Abramsky 2007] introduces “planar counterpart of $\mathsf{Int}(\mathsf{PInj})$” and observes its monoids of endomorphisms already exist in knot theory (Kauffman monoids / Temperley-Lieb algebras)

$\Rightarrow$ a 2006 talk by Hines proposes looking at “planar two-way automata” (but without characterizing their computational power…)

Planar behaviours: this drawing has no crossed edges

$$\begin{tikzcd}
q_1^{\rightarrow} \arrow[r] & q_1^{\rightarrow} \arrow[r] & q_1^{\rightarrow} \arrow[r] & q_1^{\rightarrow} \arrow[d, bend left] & q_1^{\rightarrow} \\
q_2^{\leftarrow} & q_2^{\leftarrow} \arrow[dl] & q_2^{\leftarrow} \arrow[dl] & q_2^{\leftarrow} \arrow[dl] & q_2^{\leftarrow} \arrow[dl] \\
q_3^{\leftarrow} & q_3^{\leftarrow} & \leftarrow q_3^{\leftarrow} & q_3^{\leftarrow} & q_3^{\leftarrow}
\end{tikzcd}$$

Planar behaviours: this drawing has no crossed edges



Formally: for each of these 4 behaviors, the cyclic order

$$q_1^{\text{left}} \prec q_2^{\text{left}} \prec q_3^{\text{left}} \prec q_3^{\text{right}} \prec q_2^{\text{right}} \prec q_1^{\text{right}} \prec q_1^{\text{left}}$$

does not contain any sub-cyclic-order $x \prec y \prec z \prec w \prec x$ such that

- x and z are connected by an edge (either $x \to z$ or $z \to x$)
- and y and w are also connected by an edge

$\longrightarrow$ depends on the choice of total order $q_1 < q_2 < q_3$

Expressive power of planar two-way automata

Theorem (N. & Pradic, very soon on arXiv!)

Let $L \subseteq \Sigma^*$. The following are equivalent:

- *L is a* star-free *language.*
- *L is recognized by some* planar deterministic *two-way automaton.*
- *L is recognized by some* planar reversible *two-way automaton.*

Expressive power of planar two-way automata

Theorem (N. & Pradic, very soon on arXiv!)

Let $L \subseteq \Sigma^*$. The following are equivalent:

- *L is a* star-free *language.*
- *L is recognized by some* planar deterministic *two-way automaton.*
- *L is recognized by some* planar reversible *two-way automaton.*

Comes from the connection with non-commutative linear λ-calculus:
we had used it previously to characterize star-free languages

Implicit automata in typed λ-calculi I, 2020

Expressive power of planar two-way automata

Theorem (N. & Pradic, very soon on arXiv!)

Let $L \subseteq \Sigma^*$. The following are equivalent:

- *L is a* star-free *language.*
- *L is recognized by some* planar deterministic *two-way automaton.*
- *L is recognized by some* planar reversible *two-way automaton.*

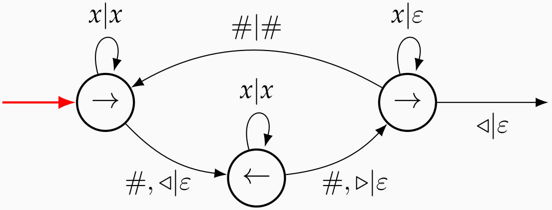
Comes from the connection with non-commutative linear λ-calculus: we had used it previously to characterize star-free languages

Implicit automata in typed λ-calculi I, 2020

Next: extensions of two-way automata

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



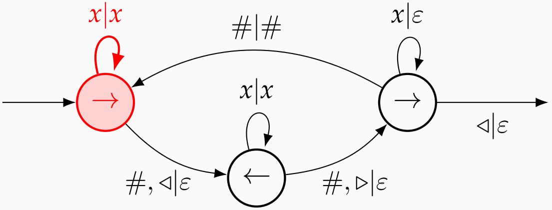
$(x \in \{a, b, c\})$

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

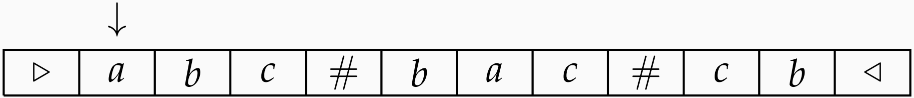
Output:

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



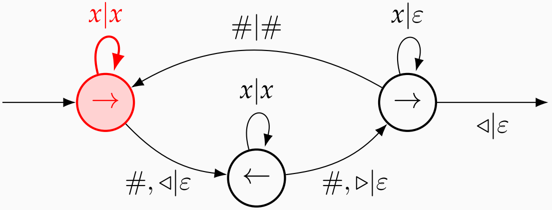
$(x \in \{a, b, c\})$



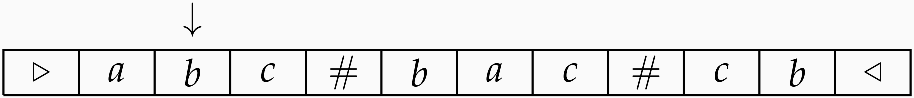
Output:

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



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

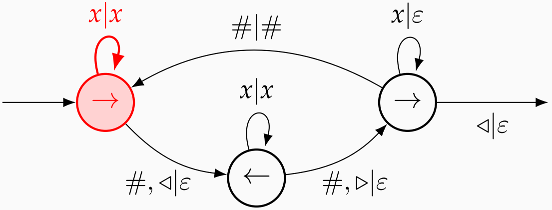


$\triangleright$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\triangleleft$

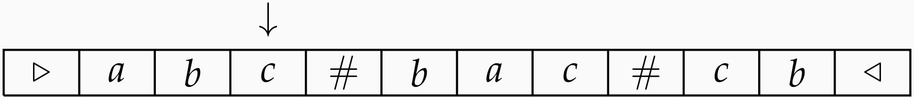
Output: a

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



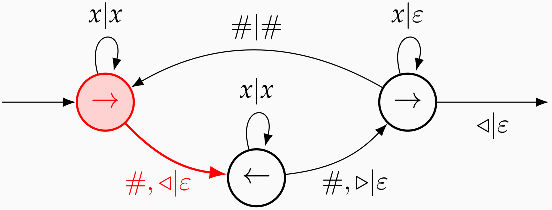
$(x \in \{a, b, c\})$



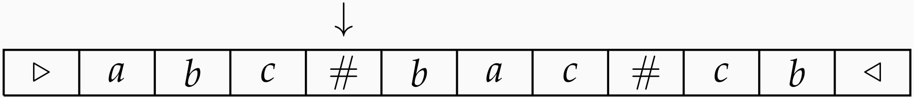
Output: ab

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



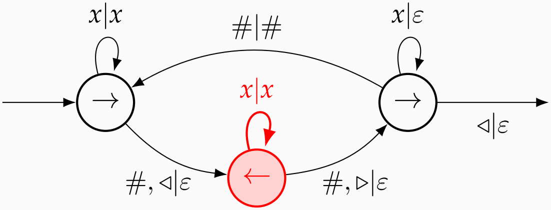
$(x \in \{a, b, c\})$



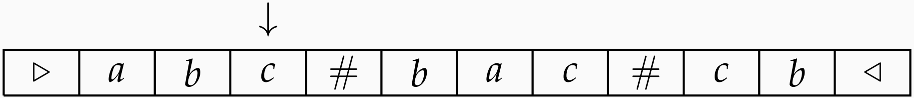
Output: abc

Deterministic two-way transducers compute regular functions (cf. Cécilia’s talk)

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



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

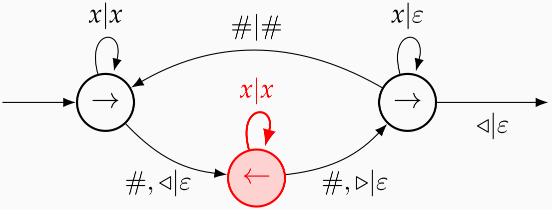


$\triangleright$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\triangleleft$

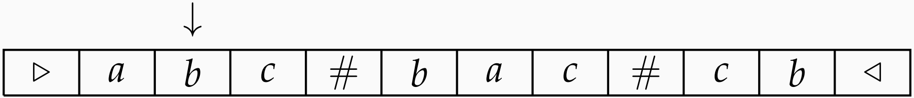
Output: abc

Deterministic two-way transducers compute regular functions (cf. Cécilia’s talk)

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



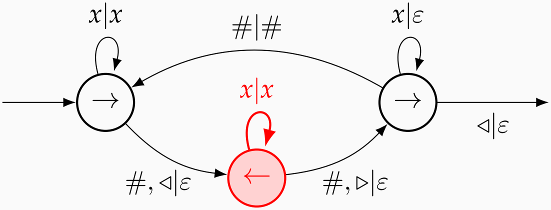
$(x \in \{a,b,c\})$



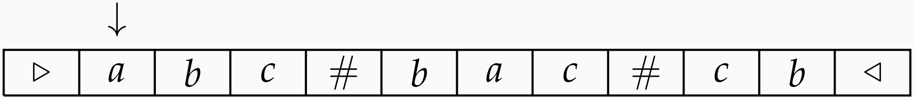
Output: $abcc$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



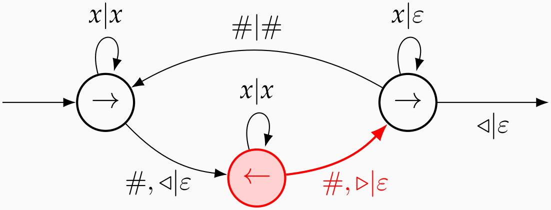
$(x \in \{a, b, c\})$



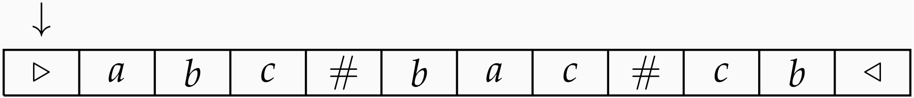
Output: *abccb*

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



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

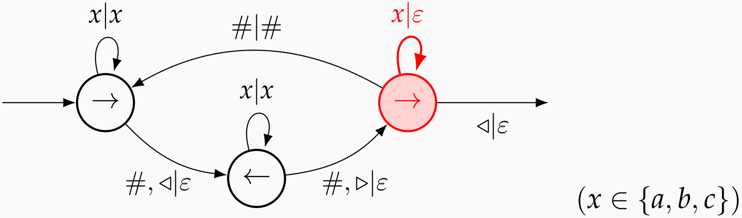


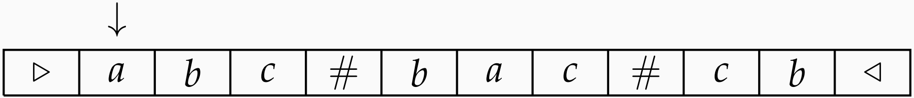
$\rhd$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\lhd$

Output: *abccba*

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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

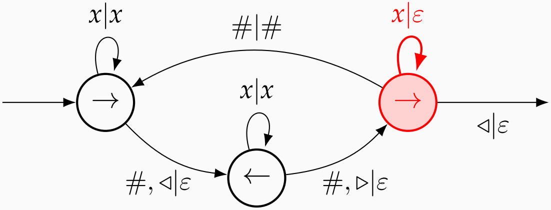




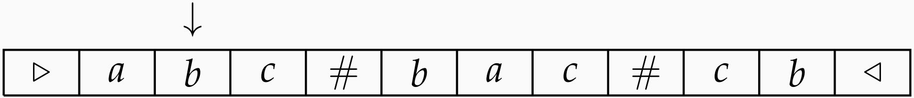
Output: *abccba*

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



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

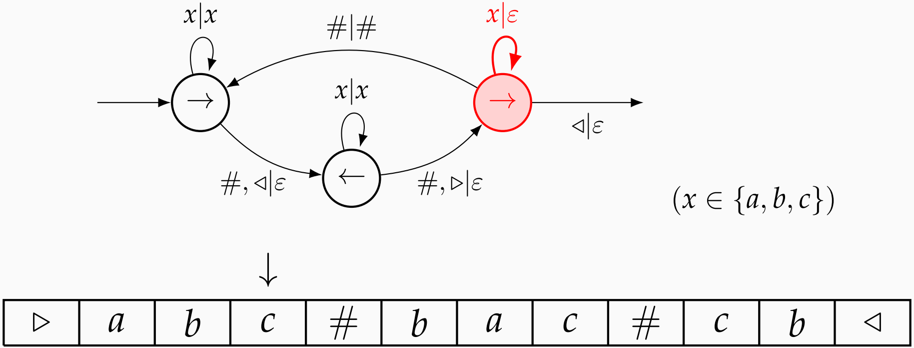


$\triangleright$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\triangleleft$

Output: *abccba*

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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

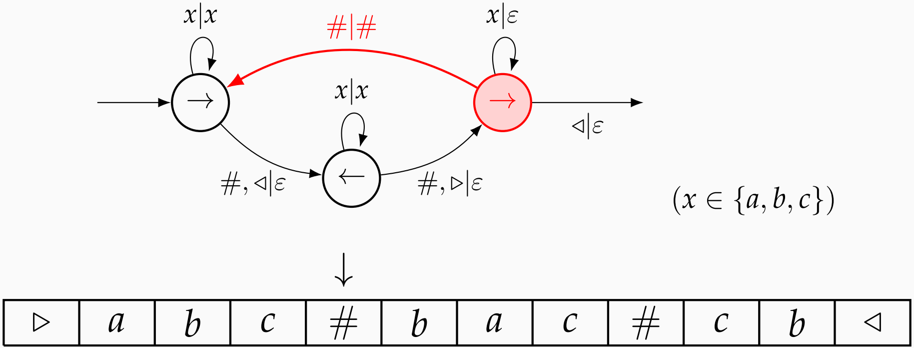


$(x \in \{a, b, c\})$

Output: *abccba*

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

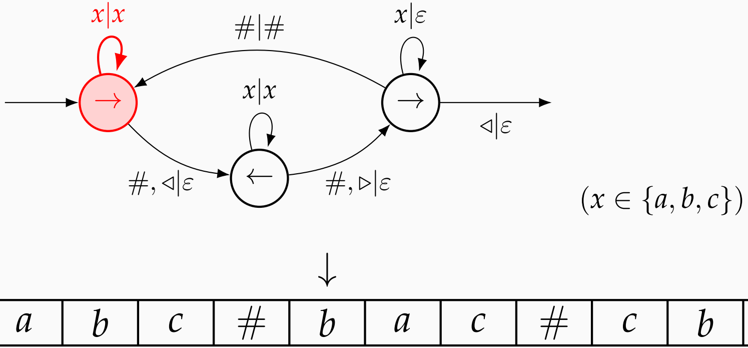
Example: $w_1\# \ldots \# w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\# \ldots \# w_n \cdot \texttt{reverse}(w_n)$



Output: *abccba*

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

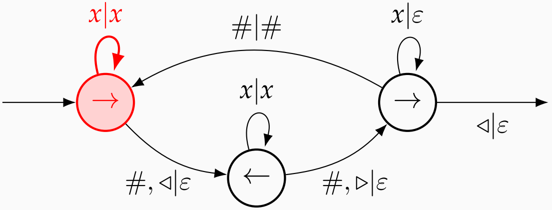
Example: $w_1\# \dots \# w_n \longmapsto w_1 \cdot \mathtt{reverse}(w_1)\# \dots \# w_n \cdot \mathtt{reverse}(w_n)$



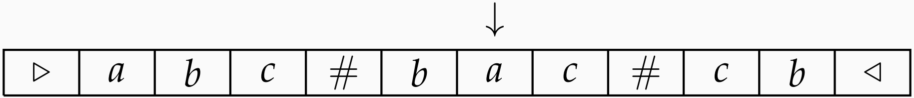
Output: $abccba\#$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



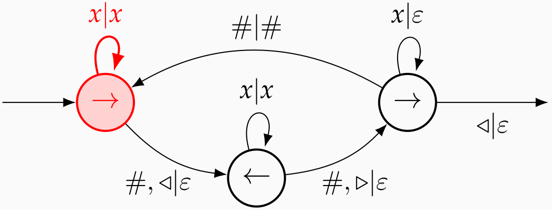
$(x\in\{a,b,c\})$



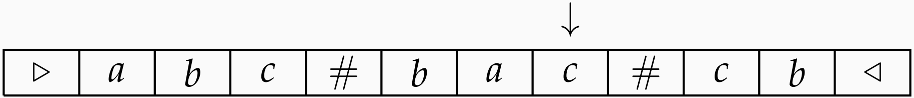
Output: $abccba\#b$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



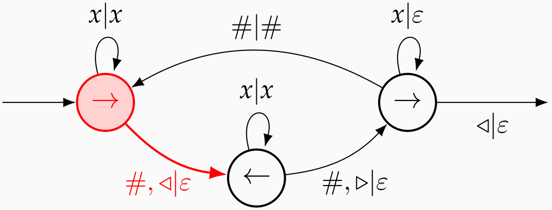
$(x \in \{a,b,c\})$



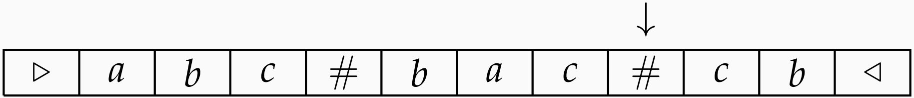
Output: $abccba\#ba$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



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

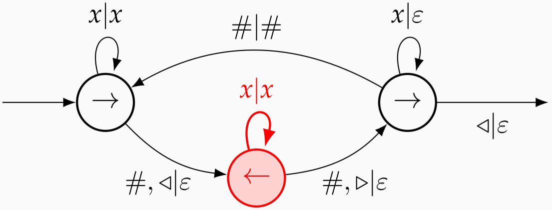


$\rhd$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\lhd$

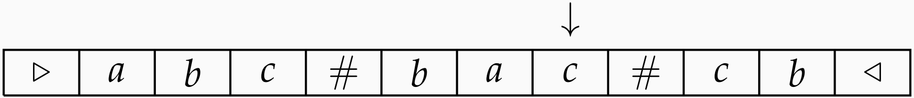
Output: $abccba\#bac$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



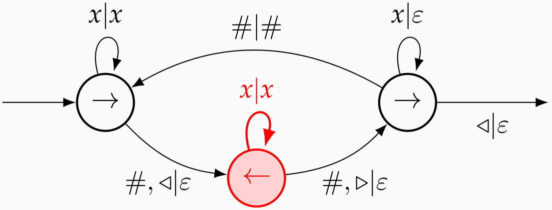
$(x \in \{a, b, c\})$



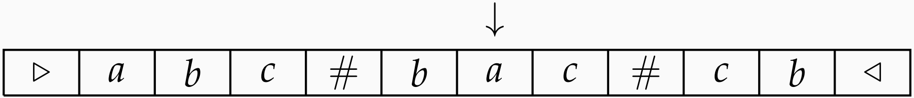
Output: $abccba\#bac$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



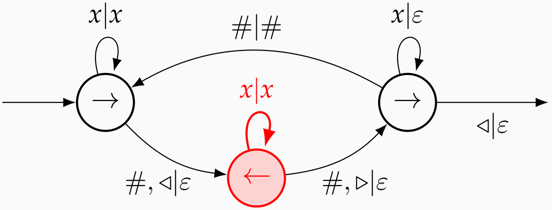
$(x \in \{a,b,c\})$



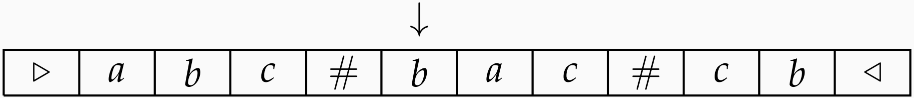
Output: $abccba\#bacc$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



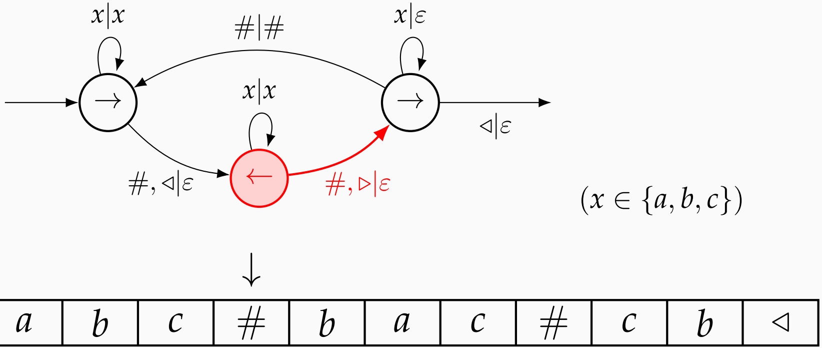
$(x \in \{a, b, c\})$



Output: $abccba\#bacca$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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

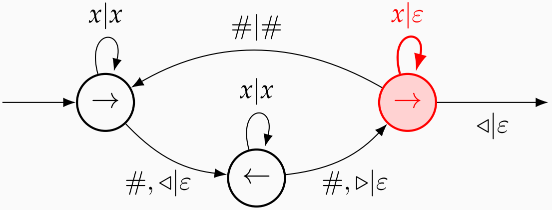


$(x \in \{a,b,c\})$

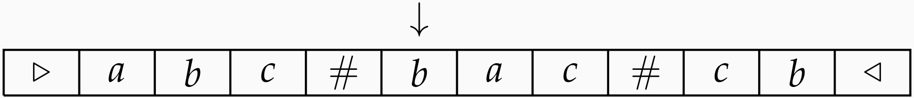
Output: $abccba\#baccab$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



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

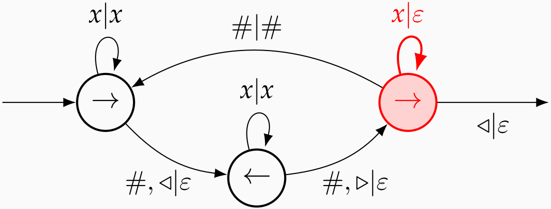


$\rhd$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\lhd$

Output: $abccba\#baccab$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



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

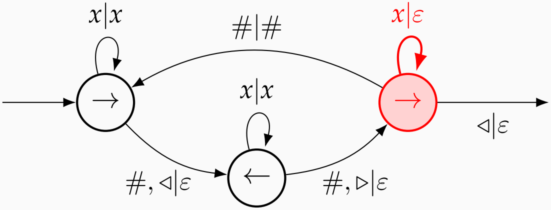
$\downarrow$

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

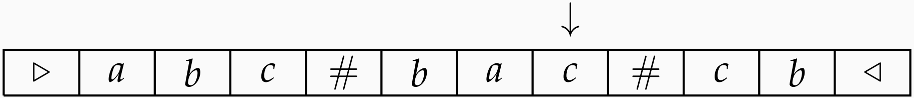
Output: $abccba\#baccab$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



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

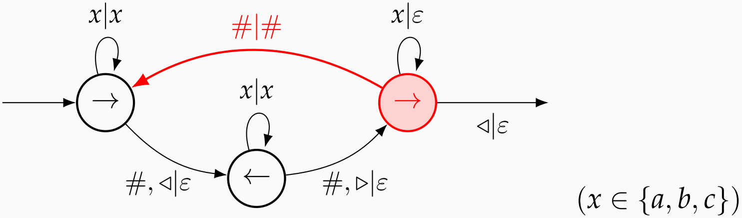


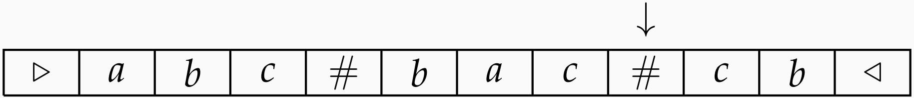
$\triangleright$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\triangleleft$

Output: $abccba\#baccab$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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

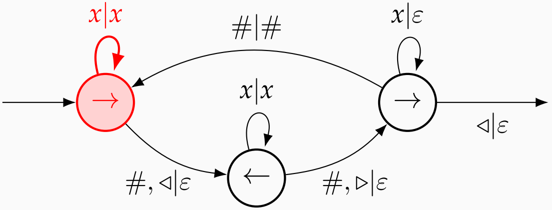




Output: *abccba#baccab*

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



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

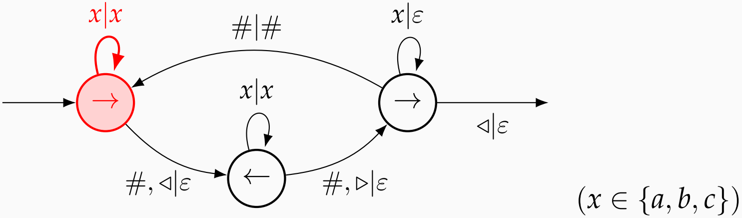
$\downarrow$

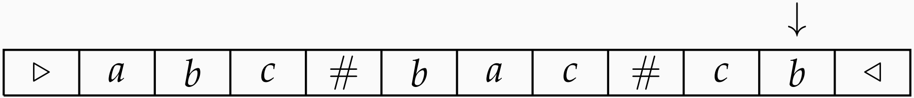
$\triangleright$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\triangleleft$

Output: $abccba\#baccab\#$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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

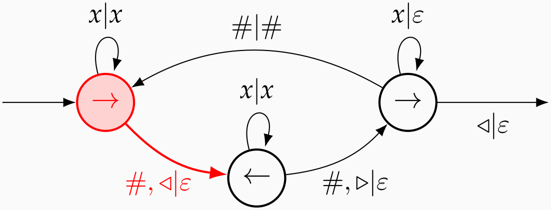




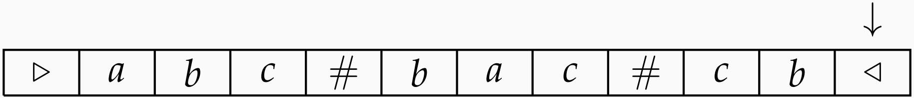
Output: $abccba\#baccab\#c$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



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

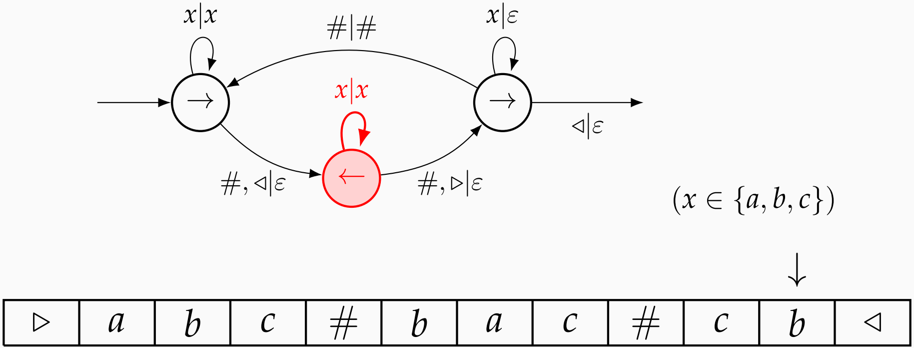


$\triangleright$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\triangleleft$

Output: $abccba\#baccab\#cb$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

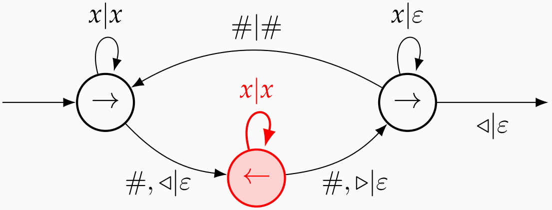
Example: $w_1\#\ldots\#w_n \longmapsto w_1 \cdot \texttt{reverse}(w_1)\#\ldots\#w_n \cdot \texttt{reverse}(w_n)$



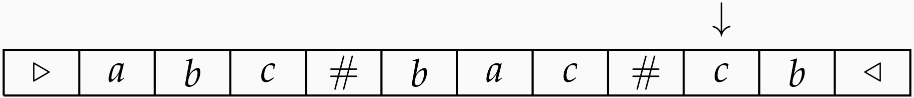
Output: $abccba\#baccab\#cb$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



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

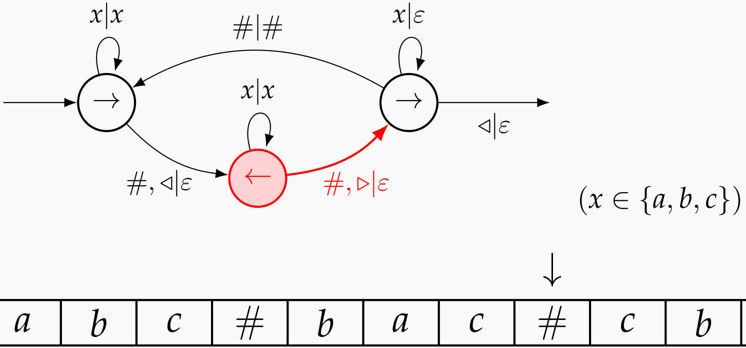


$\triangleright$	a	b	c	$\#$	b	a	c	$\#$	c	b	$\triangleleft$

Output: $abccba\#baccab\#cbb$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

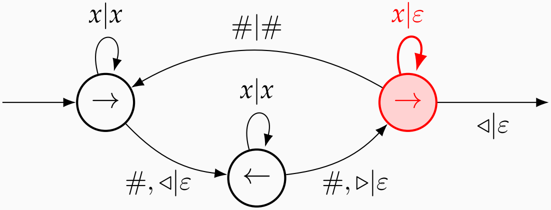
Example: $w_1\#\dots\#w_n \longmapsto w_1\cdot\mathtt{reverse}(w_1)\#\dots\#w_n\cdot\mathtt{reverse}(w_n)$



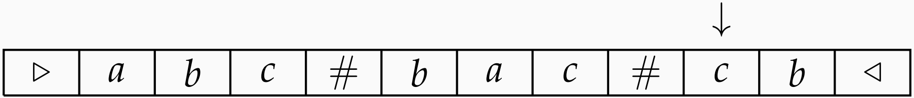
Output: *abccba#baccab#cbbc*

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



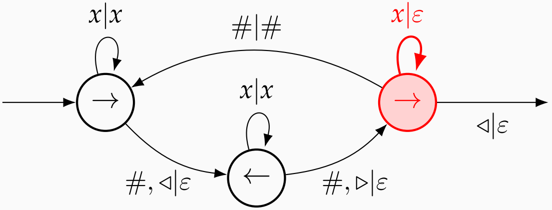
$(x\in\{a,b,c\})$



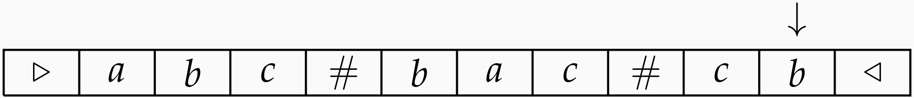
Output: $abccba\#baccab\#cbbc$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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



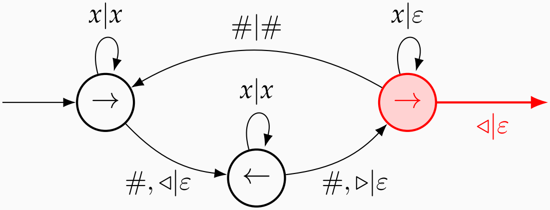
$(x \in \{a, b, c\})$



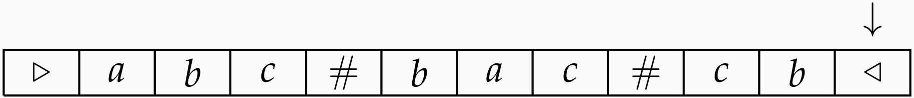
Output: $abccba\#baccab\#cbbc$

Deterministic two-way transducers compute regular functions (cf. Cécilia's talk)

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: $abccba\#baccab\#cbbc$

Extensions of two-way automata

- Two-way *transducers*: $\mathsf{Int}(\mathsf{PSet}_{\Sigma^* \times (-)})$ (Kleisli category of writer monad)
- *Tree automata* over the monoidal category $\mathsf{Int}(\mathsf{PSet})$ = *tree-walking automata*
- Both extensions together: *tree-walking transducers*

Extensions of two-way automata

- Two-way *transducers*: $\mathsf{Int}(\mathsf{PSet}_{\Sigma^* \times (-)})$ (Kleisli category of writer monad)
- *Tree automata* over the monoidal category $\mathsf{Int}(\mathsf{PSet})$ = *tree-walking automata*
- Both extensions together: *tree-walking transducers*
  ... which "are essentially attribute grammars" [Engelfriet & Maneth 2003]

Connection between attribute grammars and the Int-construction: [Katsumata 2008]

Extensions of two-way automata

- Two-way *transducers*: $\mathsf{Int}(\mathsf{PSet}_{\Sigma^* \times (-)})$ (Kleisli category of writer monad)
- *Tree automata* over the monoidal category $\mathsf{Int}(\mathsf{PSet})$ = *tree-walking automata*
- Both extensions together: *tree-walking transducers*

  ... which "are essentially attribute grammars" [Engelfriet & Maneth 2003]

Connection between attribute grammars and the Int-construction: [Katsumata 2008]

Noam's "spliced arrow operad" $\approx$ a special case of $\mathsf{Int}(\mathsf{PSet}_{\Sigma^* \times (-)})$ over trees...

output string lang. of tree-walking transd. = *multiple context-free* languages

Applications to typed λ-calculi

Could Pradic and I use semantic evaluation in $\mathsf{Int}(\mathsf{PSet}_{\Sigma^* \times (-)})$ to prove our "implicit automata" characterizations of regular (tree) functions?

Applications to typed λ-calculi

Could Pradic and I use semantic evaluation in $\mathsf{Int}(\mathsf{PSet}_{\Sigma^* \times (-)})$ to prove our "implicit automata" characterizations of regular (tree) functions? Drawbacks:

- not affine

- no additive connectives $\&/\oplus$

Applications to typed λ-calculi

Could Pradic and I use semantic evaluation in $\mathsf{Int}(\mathsf{PSet}_{\Sigma^* \times (-)})$ to prove our "implicit automata" characterizations of regular (tree) functions? Drawbacks:

- not affine: superficial issue, just use the Interaction Abstract Machine instead

  the token-based GoI mentioned at the beginning – ongoing work with Vanoni
- no additive connectives $\&/\oplus$

Applications to typed λ-calculi

Could Pradic and I use semantic evaluation in $\mathsf{Int}(\mathsf{PSet}_{\Sigma^* \times (-)})$ to prove our "implicit automata" characterizations of regular (tree) functions? Drawbacks:

- not affine: superficial issue, just use the Interaction Abstract Machine instead

  the token-based GoI mentioned at the beginning – ongoing work with Vanoni
- no additive connectives $\&/\oplus$: a feature rather than a bug, allows proving

  $\mathsf{Tree}_\Sigma[A] \multimap \mathsf{Bool}$ w/o $\&/\oplus \subseteq$ tree-walking automata $\underbrace{\subsetneq}_{\text{[Bojańczyk \& Colcombet 2005]}}$ reg. tree languages

Applications to typed λ-calculi

Could Pradic and I use semantic evaluation in $\mathsf{Int}(\mathsf{PSet}_{\Sigma^* \times (-)})$ to prove our "implicit automata" characterizations of regular (tree) functions? Drawbacks:

- not affine: superficial issue, just use the Interaction Abstract Machine instead

  the token-based GoI mentioned at the beginning – ongoing work with Vanoni
- no additive connectives $\&/\oplus$: a feature rather than a bug, allows proving

  $\mathsf{Tree}_\Sigma[A] \multimap \mathsf{Bool}$ w/o $\&/\oplus \subseteq$ tree-walking automata $\underbrace{\subsetneq}_{\text{[Bojańczyk \& Colcombet 2005]}}$ reg. tree languages

$\Rightarrow$ additive connectives *must* be included to generalize our characterization of regular functions to *trees* (but they are *incompatible* with the result on star-free languages)

Applications to typed λ-calculi

Could Pradic and I use semantic evaluation in $\mathsf{Int}(\mathsf{PSet}_{\Sigma^* \times (-)})$ to prove our "implicit automata" characterizations of regular (tree) functions? Drawbacks:

- not affine: superficial issue, just use the Interaction Abstract Machine instead

  the token-based GoI mentioned at the beginning – ongoing work with Vanoni
- no additive connectives $\&/\oplus$: a feature rather than a bug, allows proving

  $\mathsf{Tree}_\Sigma[A] \multimap \mathsf{Bool}$ w/o $\&/\oplus \subseteq$ tree-walking automata $\underbrace{\subsetneq}_{\text{[Bojańczyk \& Colcombet 2005]}}$ reg. tree languages

$\Rightarrow$ additive connectives *must* be included to generalize our characterization of regular functions to *trees* (but they are *incompatible* with the result on star-free languages)

Alternatively, in [Gallot, Lemay & Salvati 2020] – work independent from ours

"*Higher-order tree transducer*" whose memory consists of an affine λ-term; no additives, but regular lookaround ($\simeq$ preprocessing on input tree)

Final results and conclusion

Using the Interaction Abstract Machine, Vanoni and I also:

- reprove the results of [Gallot, Lemay & Salvati 2020],
  e.g. MSO transductions w/ sharing $\iff$ tree transducer using “almost affine”
  λ-terms + regular lookaround
- show that almost affine higher-order tree transducers with “!-depth 1”
  $\iff$ *invisible pebble tree transducers* [Engelfriet, Hoogeboom & Samwel 2007]

Final results and conclusion

Using the Interaction Abstract Machine, Vanoni and I also:

- reprove the results of [Gallot, Lemay & Salvati 2020], e.g. MSO transductions w/ sharing $\iff$ tree transducer using "almost affine" λ-terms + regular lookaround
- show that almost affine higher-order tree transducers with "!-depth 1" $\iff$ *invisible pebble tree transducers* [Engelfriet, Hoogeboom & Samwel 2007]

Recap of previous discussion (j.w.w. Pradic)

- Connections between $\mathsf{Int}(\mathsf{PSet})$ (categorical Geometry of Interaction), two-way automata [Hines 2003] and tree-walking transducers [Katsumata 2008]
- Leads to *planar* two-way automata, recognizing star-free languages and planar two-way transducers = aperiodic/first-order regular functions

Final results and conclusion

Thanks for your attention! Questions?

Using the Interaction Abstract Machine, Vanoni and I also:

- reprove the results of [Gallot, Lemay & Salvati 2020],
  e.g. MSO transductions w/ sharing $\iff$ tree transducer using "almost affine"
  λ-terms + regular lookaround
- show that almost affine higher-order tree transducers with "!-depth 1"
  $\iff$ *invisible pebble tree transducers* [Engelfriet, Hoogeboom & Samwel 2007]

Recap of previous discussion (j.w.w. Pradic)

- Connections between Int(PSet) (categorical Geometry of Interaction),
  two-way automata [Hines 2003] and tree-walking transducers [Katsumata 2008]
- Leads to *planar* two-way automata, recognizing star-free languages
  and planar two-way transducers = aperiodic/first-order regular functions