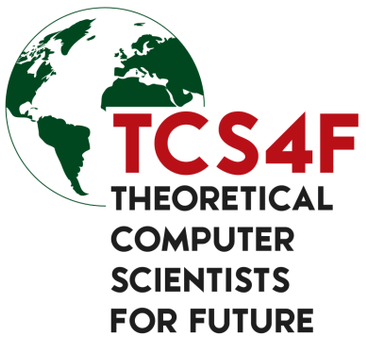
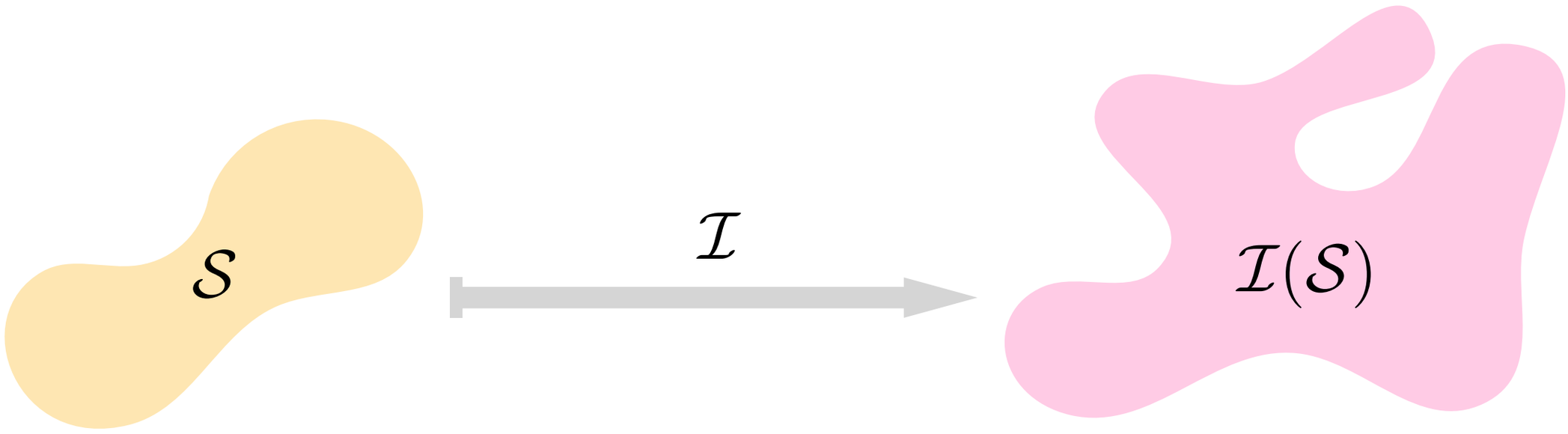
String-to-string MSO-set-interpretations

Thomas Colcombet, 30 May 2025, Finite Model Theory, Les Houches

joint work with Nathan Lhote, Lê Thành Dũng (Tito) Nguyễn and Pierre Ohlmann.



FO-interpretation of dimension k

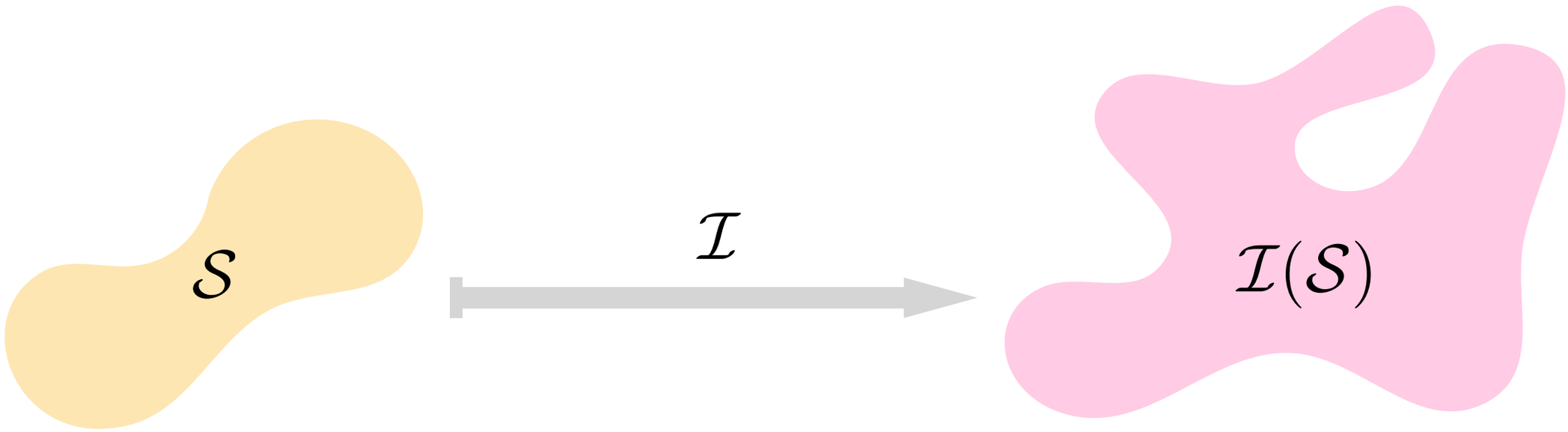


Input structure of universe U.

Output structure
Elements are k-tuples of element of S.
Relations defined in S by FO-formulas.

Eg. Path $\longmapsto$ grids.

FO-interpretation of dimension k



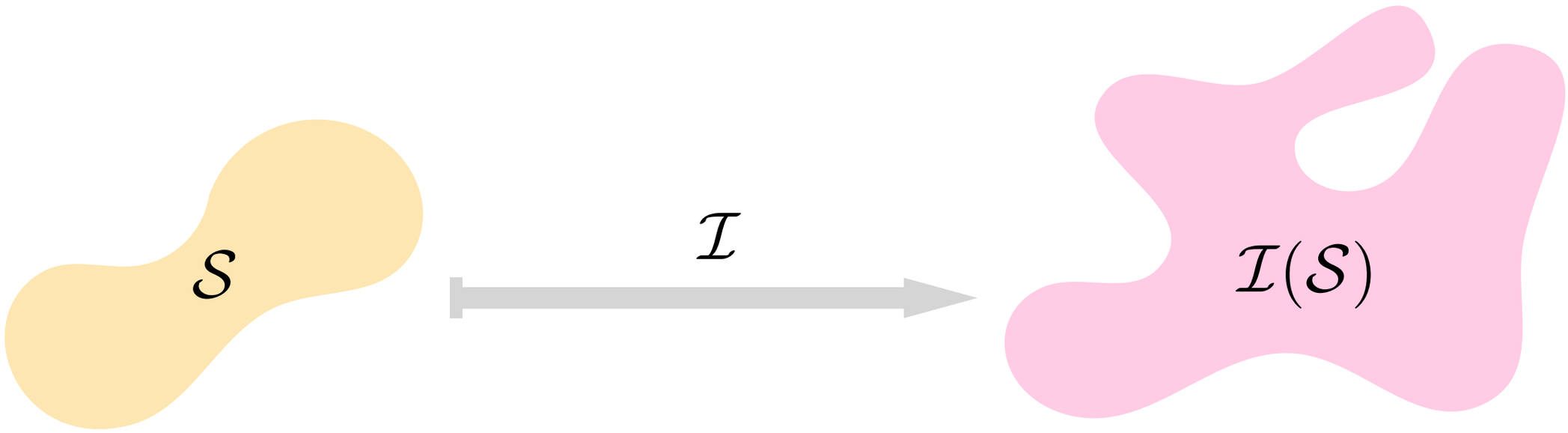
Input structure of universe U.

Output structure
Elements are k-tuples of element of S.
Relations defined in S by FO-formulas.

Eg. Path $\longmapsto$ grids.

Key fact: For all FO-sentences Φ,
there exists effectively an FO-sentence $\Phi^{\mathcal{I}}$,
such that $\mathcal{S} \models \Phi^{\mathcal{I}}$ iff $\mathcal{I}(\mathcal{S}) \models \Phi$ for all structures $\mathcal{S}$.

(parameterless) MSO-transductions
= MSO-interpretation 1-dimensional with k-copying

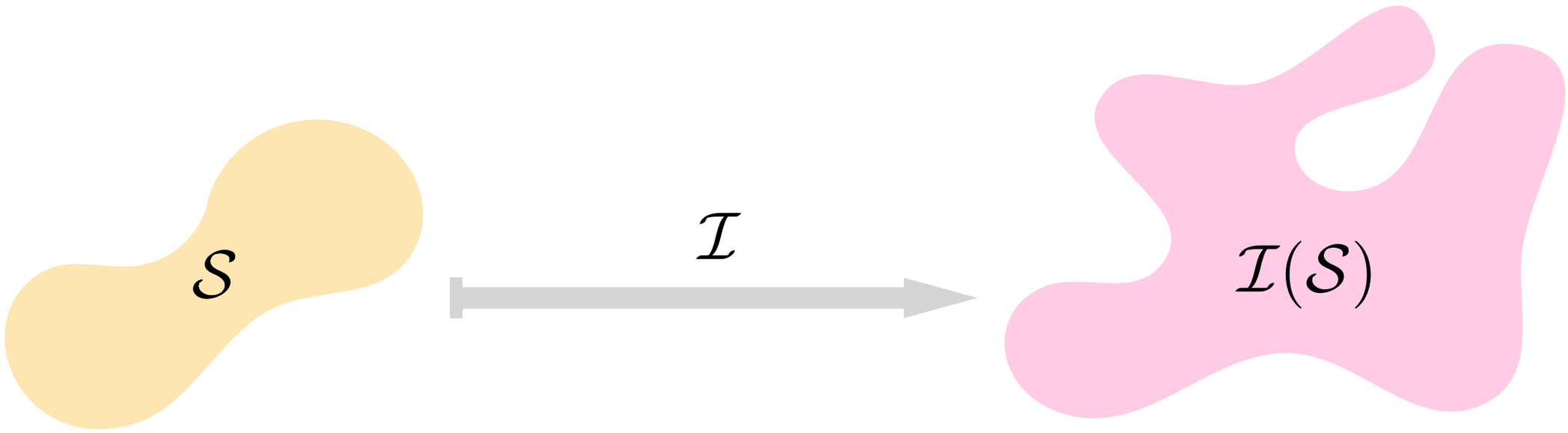


Input structure of universe U.

Output structure
Universe MSO-definable in U $\times$ [k] .
Relations defined in S by MSO-formulas.

Eg. trees $\longmapsto$ bounded clique-width.

(parameterless) MSO-transductions
= MSO-interpretation 1-dimensional with k-copying



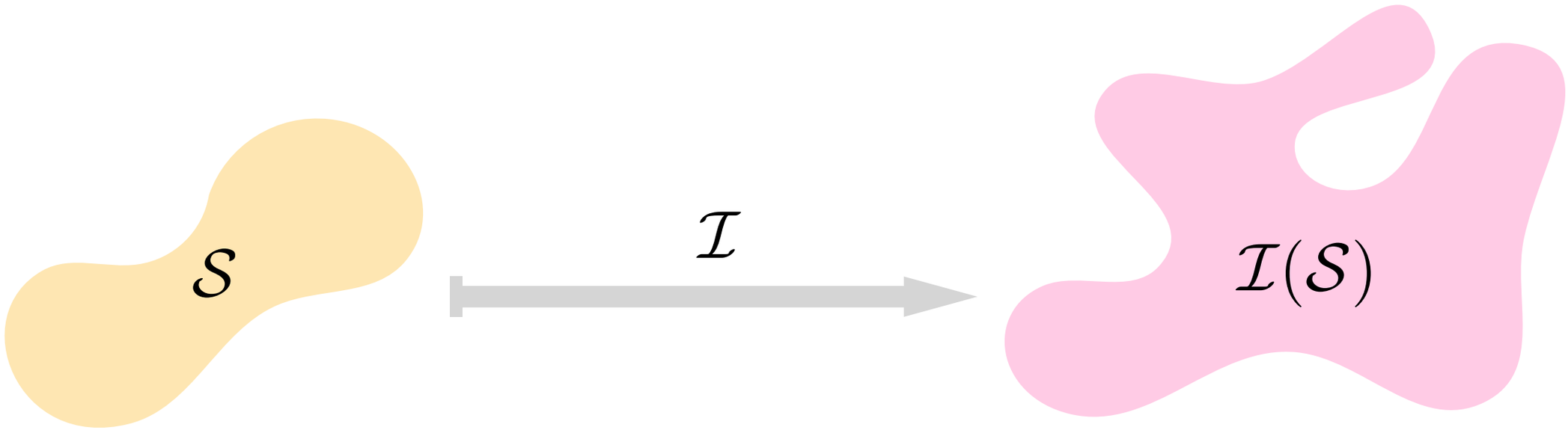
Input structure of universe U.

Output structure
Universe MSO-definable in U $\times$ [k] .
Relations defined in S by MSO-formulas.

Eg. trees $\longmapsto$ bounded clique-width.

Key fact: For all MSO-sentences Φ,
there exists effectively an MSO-sentence $\Phi^{\mathcal{I}}$,
such that $\mathcal{S} \models \Phi^{\mathcal{I}}$ iff $\mathcal{I}(\mathcal{S}) \models \Phi$ for all structures $\mathcal{S}$.

MSO-set-interpretations

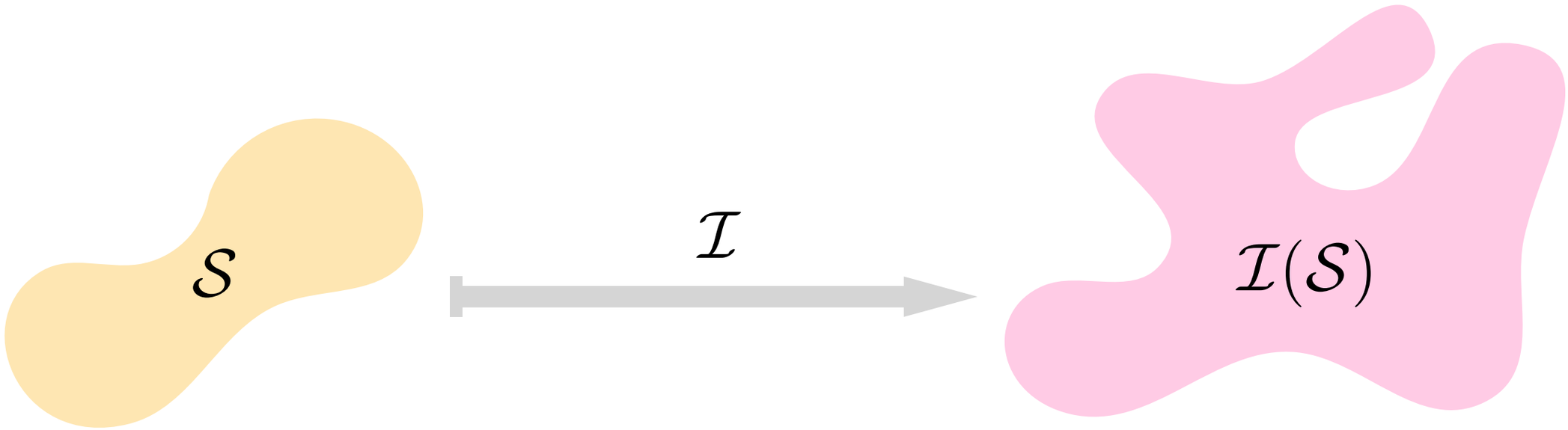


Input structure of universe U.

Output structure
Universe MSO-definable in 2^U .
Relations defined in S by MSO-formulas.

Eg. $(\omega,<) \longmapsto$ automatic structure.

MSO-set-interpretations



Input structure of universe U.

Output structure
Universe MSO-definable in 2^U .
Relations defined in S by MSO-formulas.

Eg. (ω,<) $\longmapsto$ automatic structure.

Key fact: For all FO-sentences Φ,
there exists effectively an MSO-sentence $\Phi^{\mathcal{I}}$,
such that $\mathcal{S} \models \Phi^{\mathcal{I}}$ iff $\mathcal{I}(\mathcal{S}) \models \Phi$ for all structures $\mathcal{S}$.

For FO-interpretations of dimension k	$\mathcal{U}^k$	The inverse image of FO-definable is effectively FO-definable
For MSO-transductions = MSO-interpretations of dimension 1	$\mathcal{U} \times k$	The inverse image of MSO-definable is effectively FO-definable
For MSO-set-interpretations	$(2^{\mathcal{U}})^k$	The inverse image of FO-definable is effectively MSO-definable

For FO-interpretations of dimension k	$\mathcal{U}^k$	The inverse image of FO-definable is effectively FO-definable
For MSO-transductions = MSO-interpretations of dimension 1	$\mathcal{U} \times k$	The inverse image of MSO-definable is effectively FO-definable
For MSO-set-interpretations	$(2^{\mathcal{U}})^k$	The inverse image of FO-definable is effectively MSO-definable

Theorem [Bojańczyk, Kiefer, Lhote19]:

For MSO-interpretations of dimension k from strings to strings	$\mathcal{U}^k$	The inverse image of MSO-definable is effectively MSO-definable

Finite linear orders
+ unary predicates

For FO-interpretations of dimension k	$\mathcal{U}^k$	The inverse image of FO-definable is effectively FO-definable
For MSO-transductions = MSO-interpretations of dimension 1	$\mathcal{U} \times k$	The inverse image of MSO-definable is effectively FO-definable
For MSO-set-interpretations	$(2^{\mathcal{U}})^k$	The inverse image of FO-definable is effectively MSO-definable

Theorem [Bojańczyk, Kiefer, Lhote19]:

For MSO-interpretations of dimension k from strings to strings	$\mathcal{U}^k$	The inverse image of MSO-definable is effectively MSO-definable

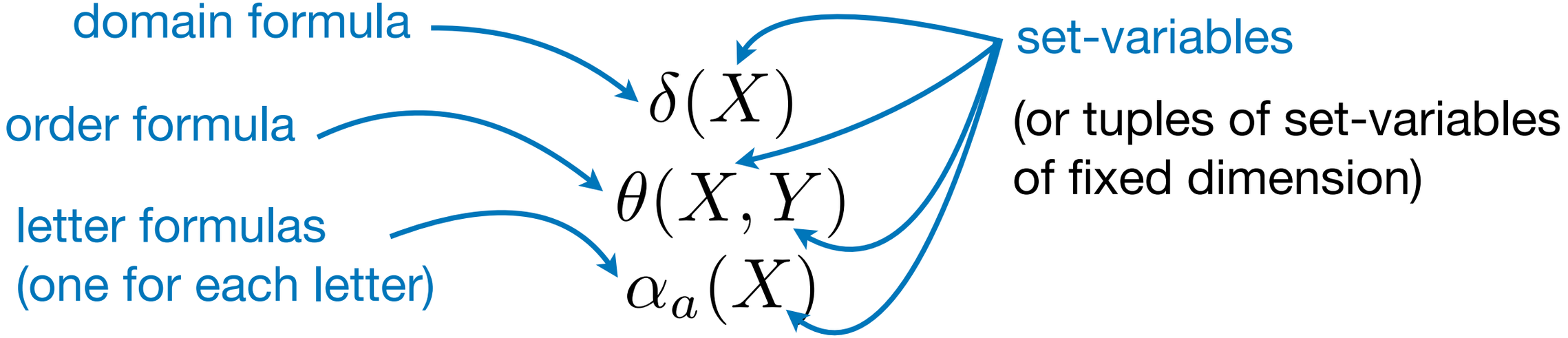
Theorem:

For MSO-set-interpretations from strings to strings	$(2^{\mathcal{U}})^k$	The inverse image of MSO-definable is effectively MSO-definable

String-to-string MSO-set-interpretation

Input and output signature: linear order symbol + unary predicates (letters)

A **string-to-string MSO-set-interpretation** consists of a tuple of MSO-formulas over the input signature:



Given an **input string**, it defines the **output string** consisting of:

- positions are subsets of the input that satisfy the domain formula
- the order is interpreted using the order formula (it is required to produce a total order!)
- the unary predicates (letters) are interpreted using the letter formulas.

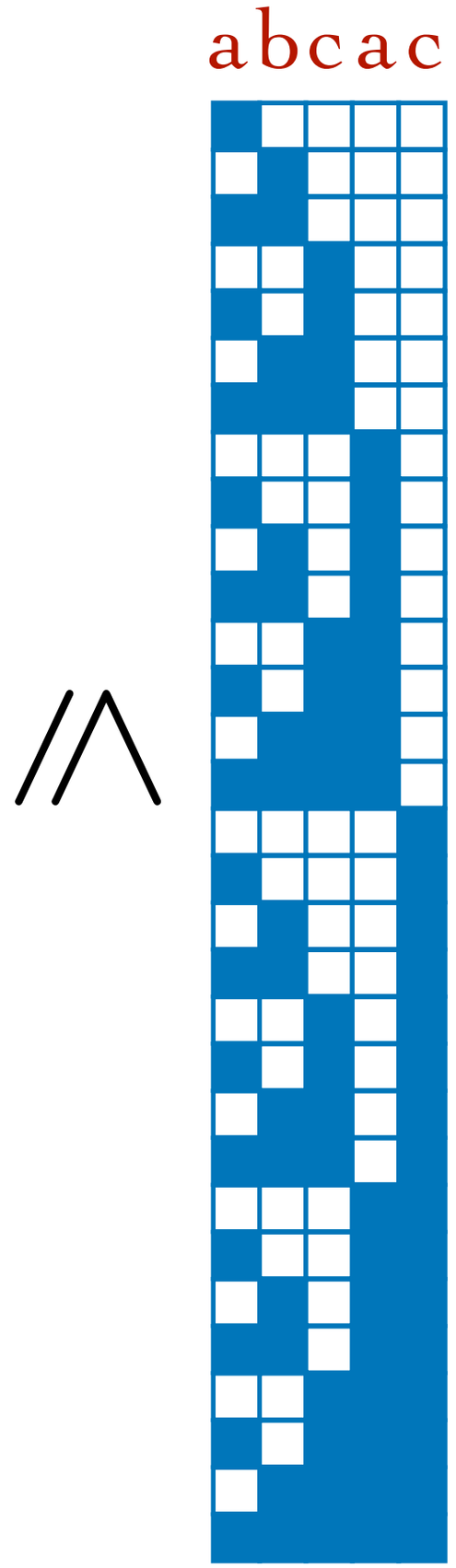
String-to-string MSO-set-interpretation

Example:

domain = non-empty sets

order = lexicographic

letter = the input letter at the maximal position in the set



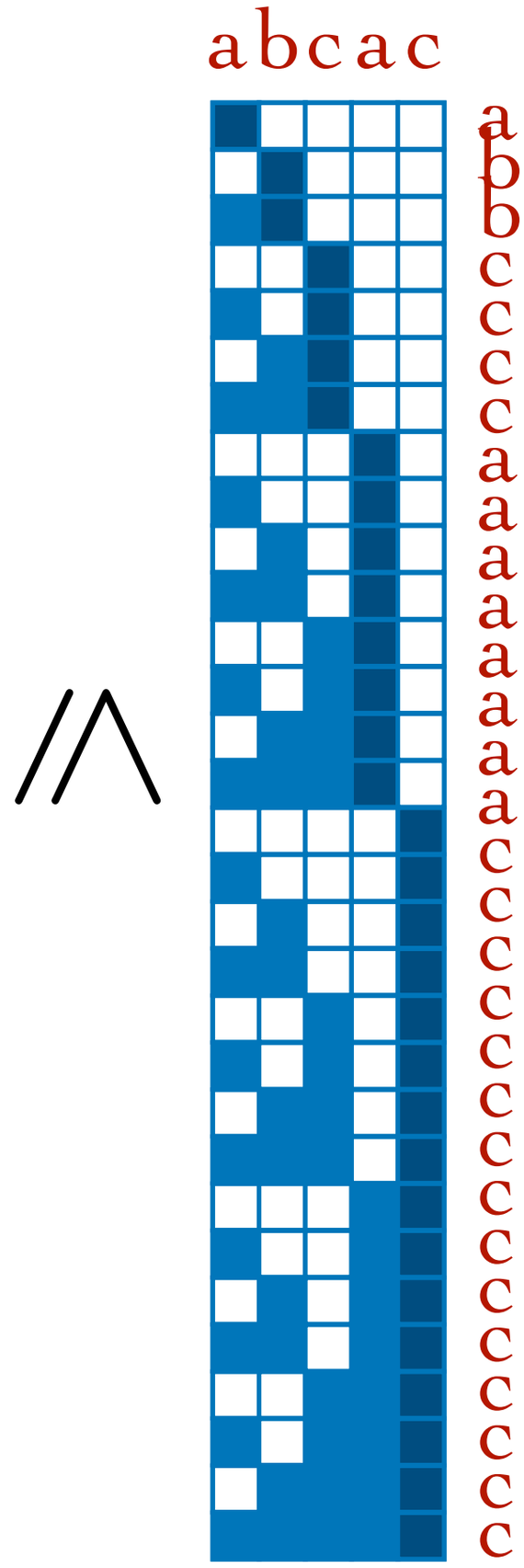
String-to-string MSO-set-interpretation

Example:

domain = non-empty sets

order = lexicographic

letter = the input letter at the maximal position in the set



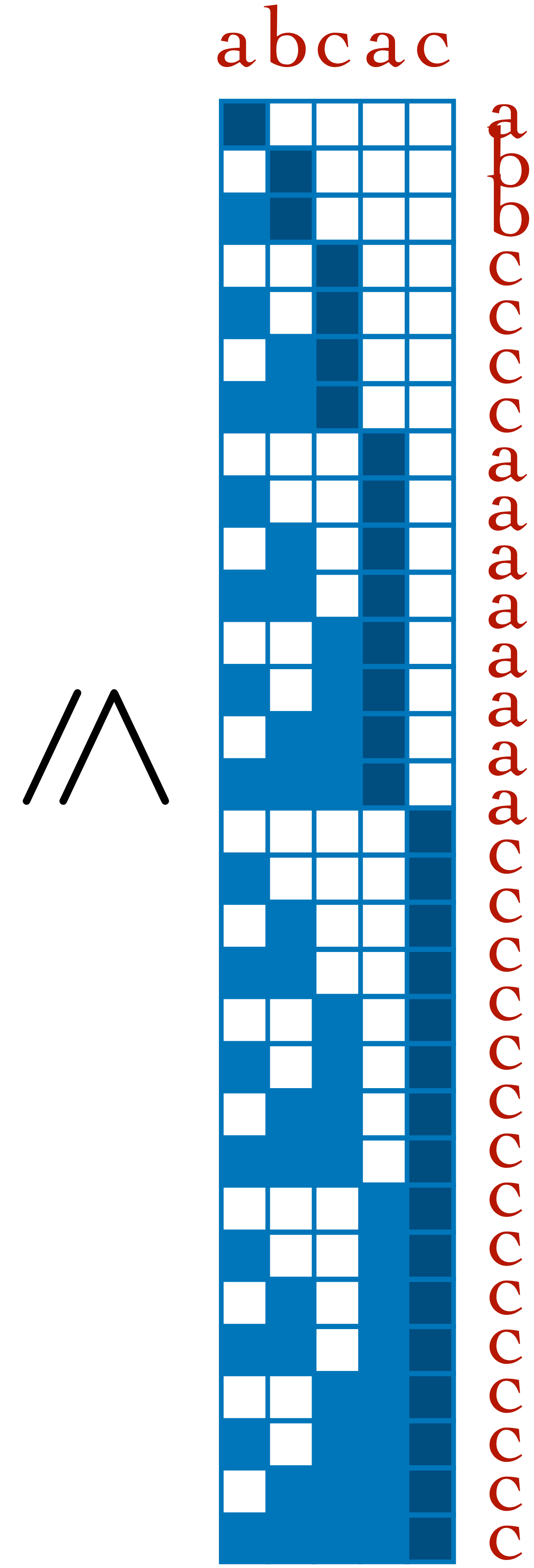
String-to-string MSO-set-interpretation

Example:

domain = non-empty sets

order = lexicographic

letter = the input letter at the maximal position in the set



$$f: \quad \Sigma^* \to \Sigma^*$$

$$a_0 a_1 \ldots a_{n-1} \mapsto a_0^{2^0} a_1^{2^1} \ldots a_{n-1}^{2^{n-1}}$$

Polyregular functions

Polyregular functions correspond to the special case where tuples of first-order variables are used instead of tuples of sets (still MSO formulas) = MSO-interpretation of dimension k.

Polyregular functions

Polyregular functions correspond to the special case where tuples of first-order variables are used instead of tuples of sets (still MSO formulas) = MSO-interpretation of dimension k.

Example:

domain = pairs of positions (x,y) with x<=y

order = lexicographic over (-x,y)

letter = the letter at y

Polyregular functions

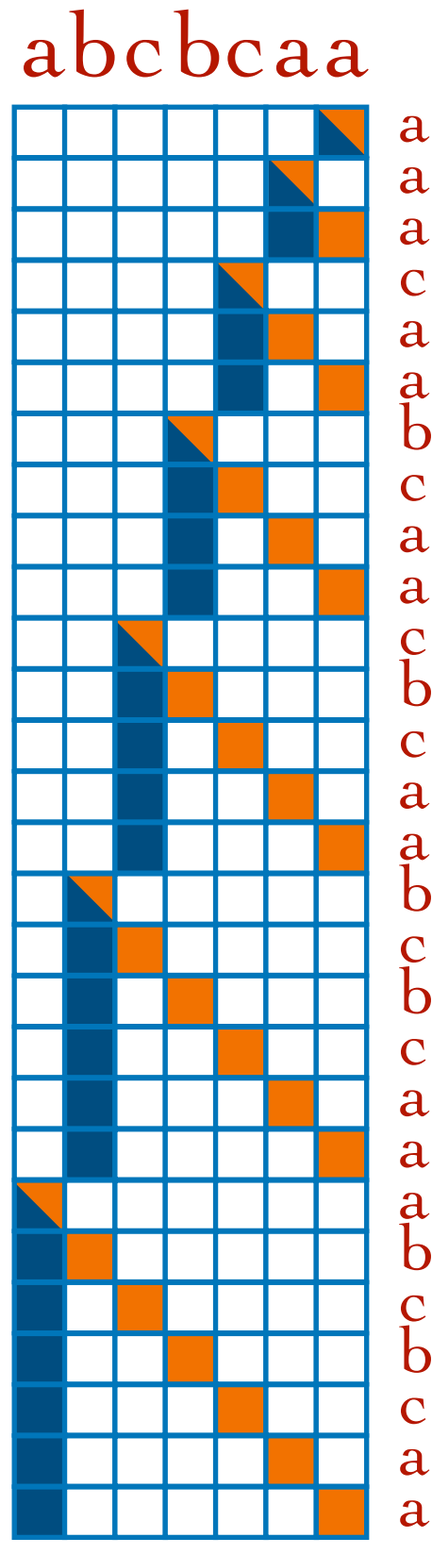
Polyregular functions correspond to the special case where tuples of first-order variables are used instead of tuples of sets (still MSO formulas) = MSO-interpretation of dimension k.

Example:

domain = pairs of positions (x,y) with x<=y

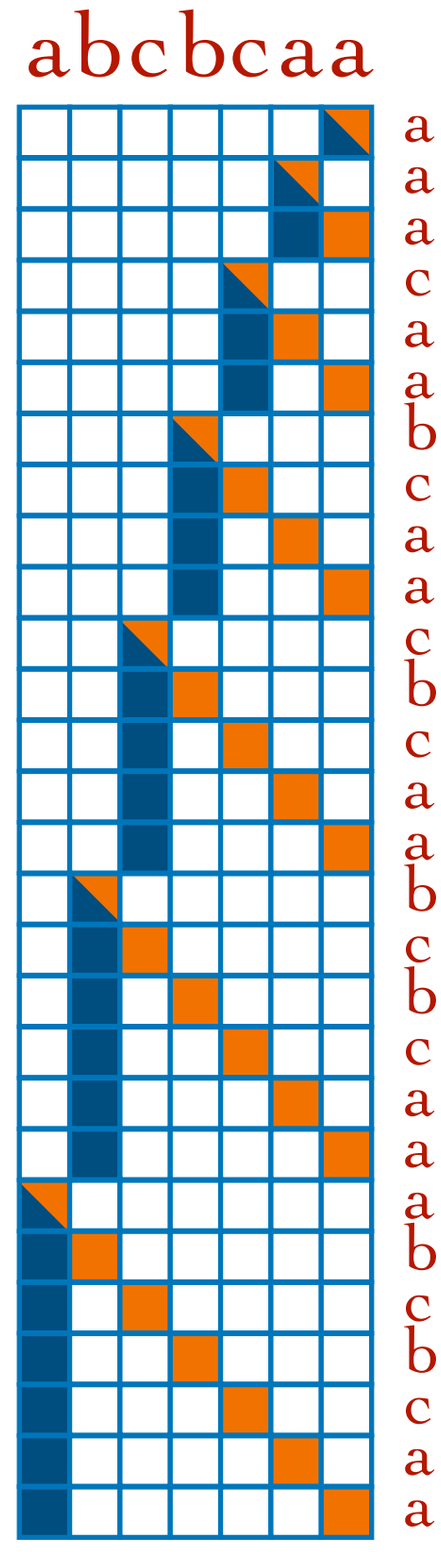
order = lexicographic over (-x,y)

letter = the letter at y



Polyregular functions

Polyregular functions correspond to the special case where tuples of first-order variables are used instead of tuples of sets (still MSO formulas) = MSO-interpretation of dimension k.



Example:

domain = pairs of positions (x,y) with x<=y

order = lexicographic over (-x,y)

letter = the letter at y

$$f: \quad \Sigma^* \quad \to \quad \Sigma^*$$

$$a_0 a_1 \dots a_{n-1} \mapsto (a_{n-1})(a_{n-2} a_{n-1}) \dots (a_0 \dots a_{n-1})$$

Main result

Theorem: For a string-to-string map, the following items are effectively equivalent:

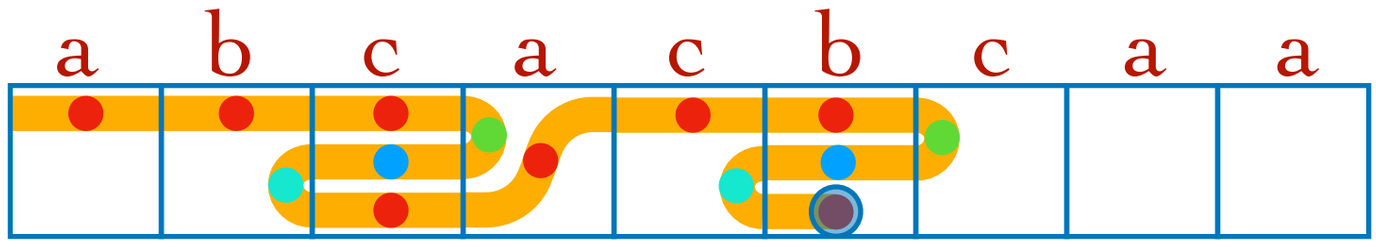
- being definable by an MSO-set-interpretation,
- being definable by an Ariadne-transducer,
- being definable by an Ariadne-transducer with regular lookaround of the configuration,
- being definable by a push-only string-to-tree Ariadne-transducer followed by a yield.

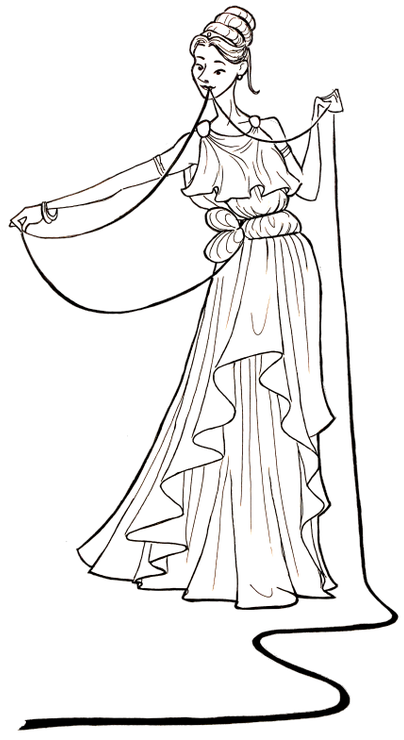
Furthermore, the inverse image of a regular string language under such a map is effectively regular.

Joséphine C.

String-to-string Ariadne-transducer

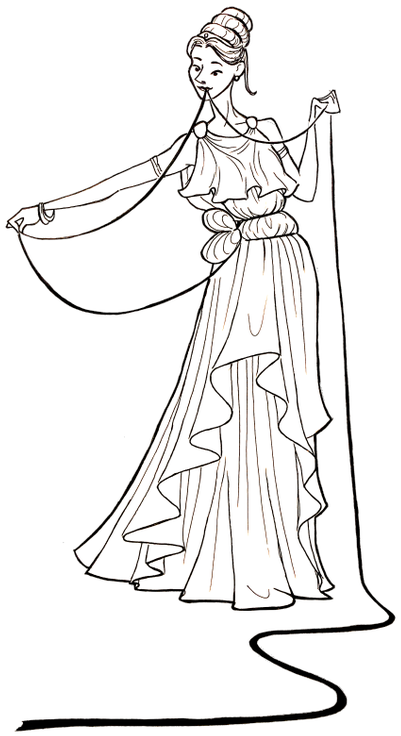
A **configuration** of an **Ariadne-transducer** is a walk on the input word, decorated with **states** (from a finite set), that can visit at most k times each position of the input string:

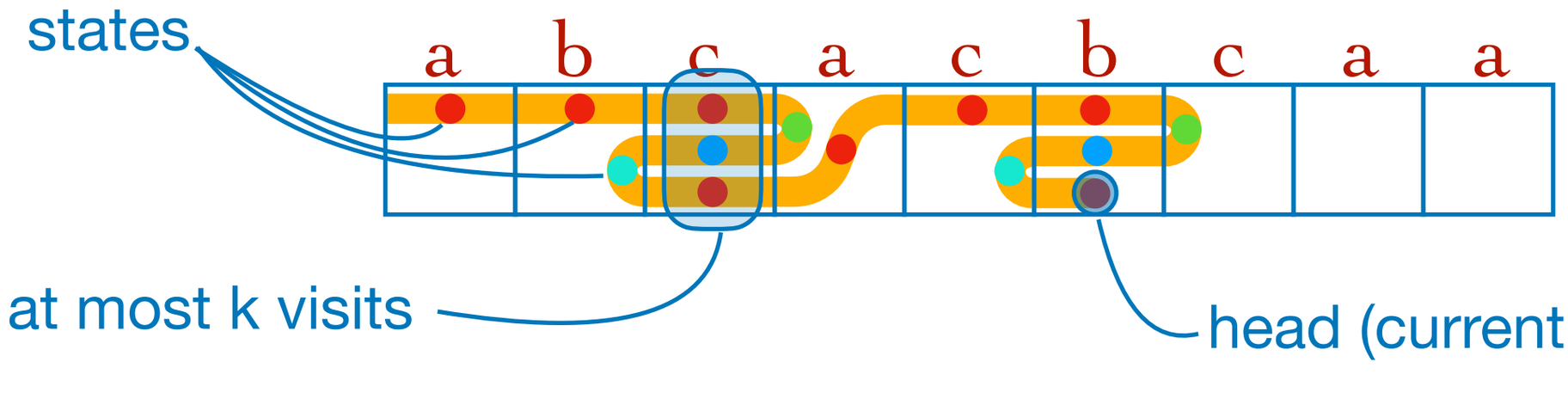




String-to-string Ariadne-transducer

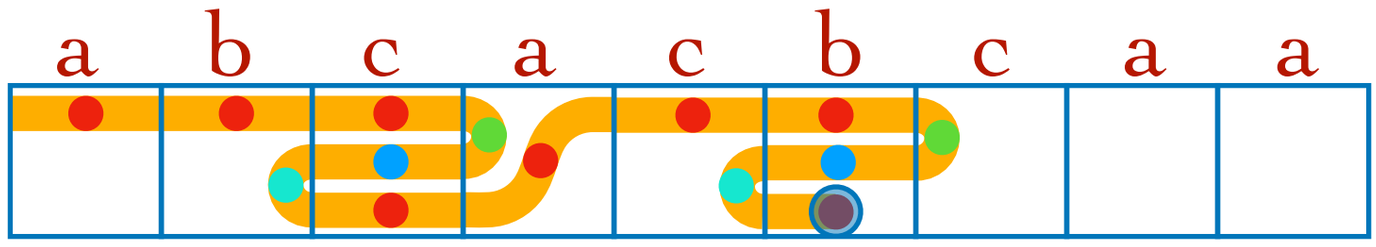
A **configuration** of an **Ariadne-transducer** is a walk on the input word, decorated with **states** (from a finite set), that can visit at most k times each position of the input string:

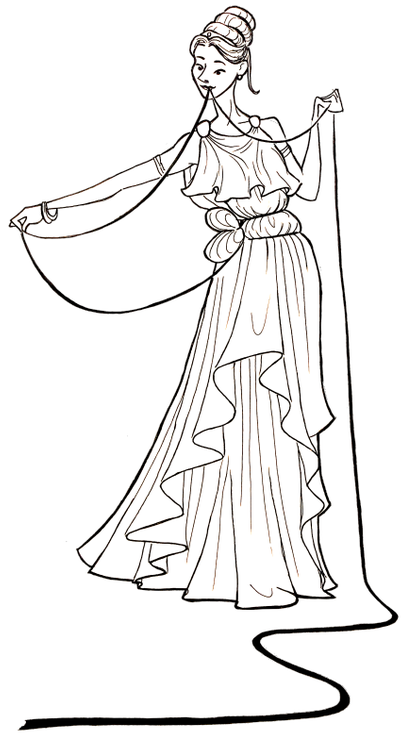




String-to-string Ariadne-transducer

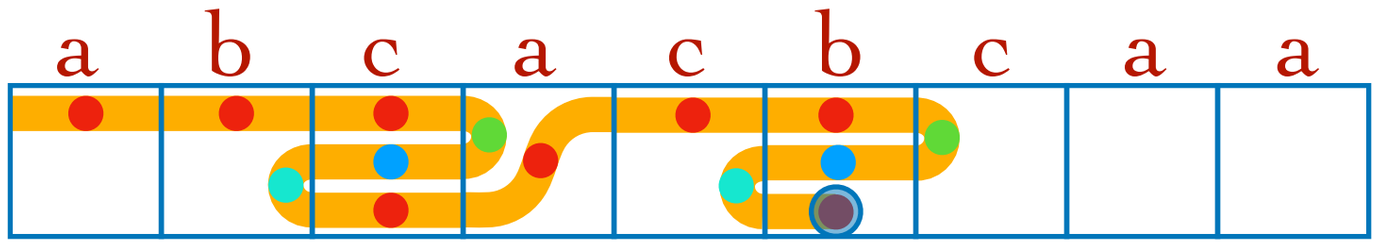
A **configuration** of an **Ariadne-transducer** is a walk on the input word, decorated with **states** (from a finite set), that can visit at most k times each position of the input string:

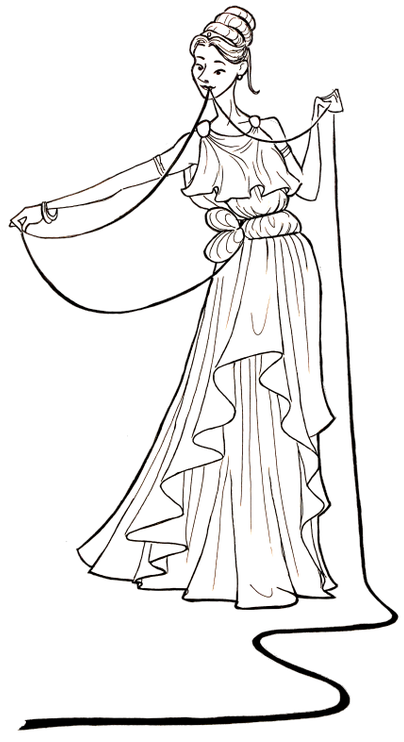




String-to-string Ariadne-transducer

A **configuration** of an **Ariadne-transducer** is a walk on the input word, decorated with **states** (from a finite set), that can visit at most k times each position of the input string:





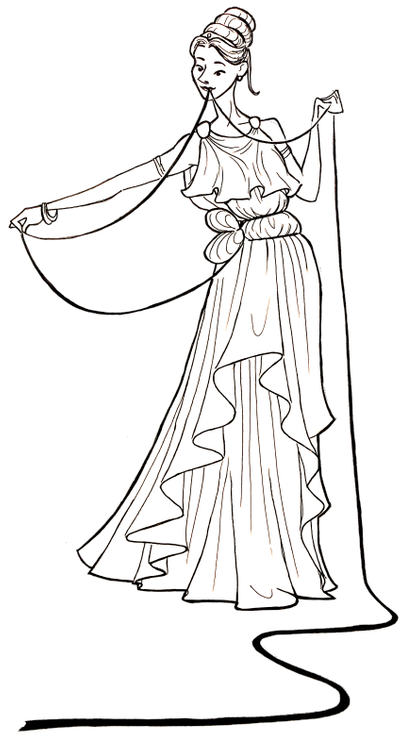
A **transition** is decided based on the full content of the configuration at the head position (input letter + states), and can possibly either

- **[push]** extend the walk of one step (left or right), or
- **[pop]** rewind the walk of one step,

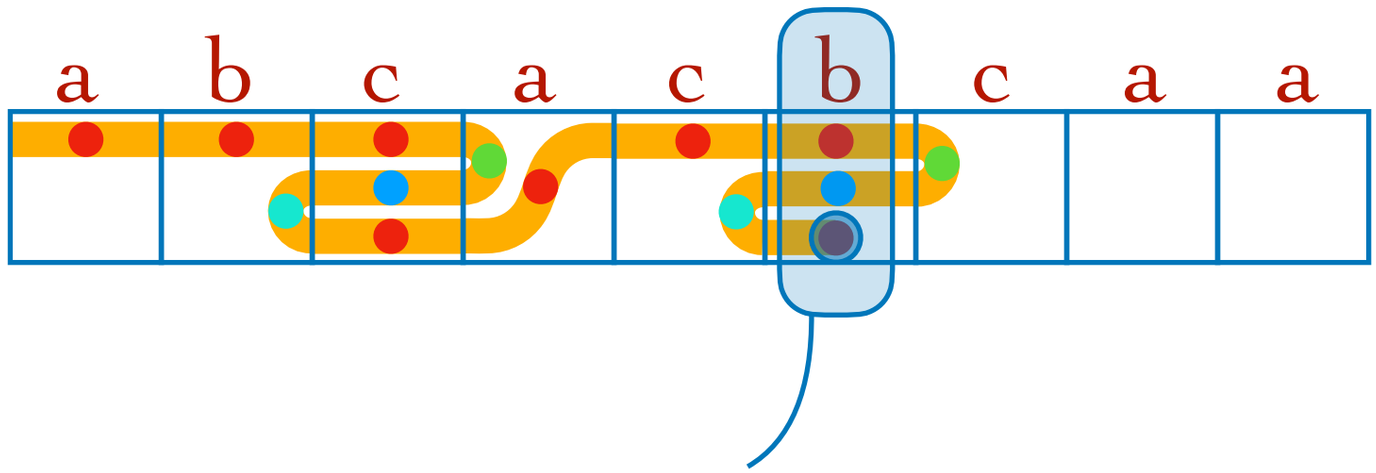
and at the same time it

- **[updates]** the head state, and
- **[outputs]** possibly produces a symbol from the output alphabet.

String-to-string Ariadne-transducer



A **configuration** of an **Ariadne-transducer** is a walk on the input word, decorated with **states** (from a finite set), that can visit at most k times each position of the input string:



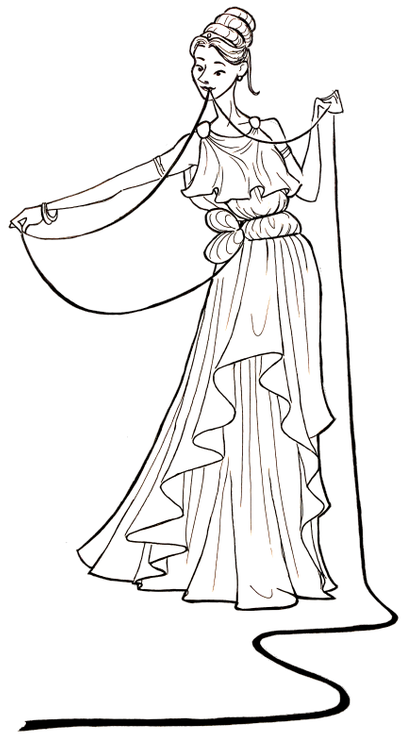
A **transition** is decided based on the full content of the configuration at the head position (input letter + states), and can possibly either

- **[push]** extend the walk of one step (left or right), or
- **[pop]** rewind the walk of one step,

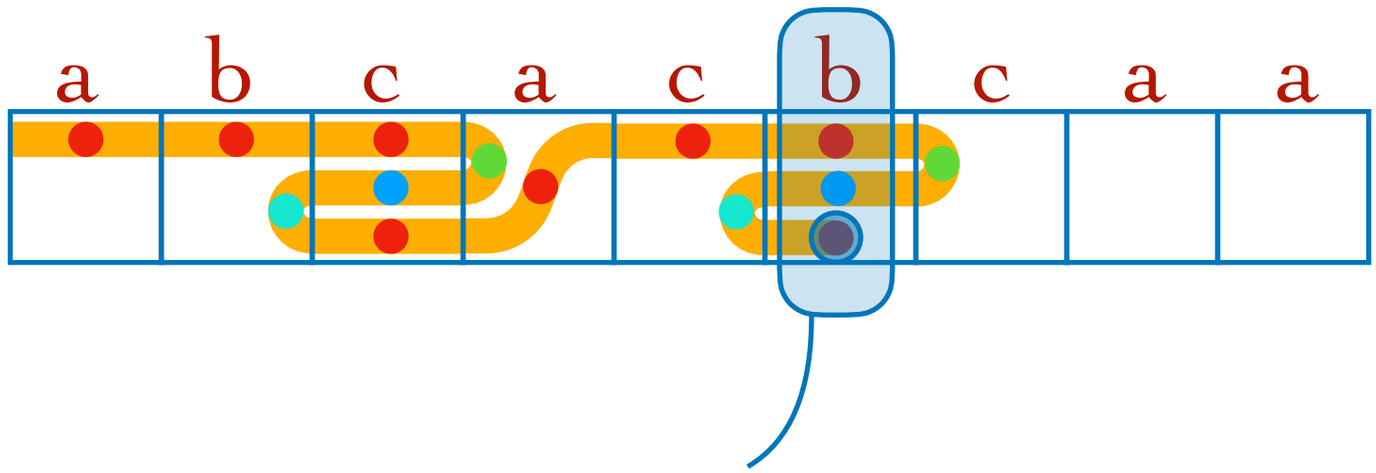
and at the same time it

- **[updates]** the head state, and
- **[outputs]** possibly produces a symbol from the output alphabet.

String-to-string Ariadne-transducer



A **configuration** of an **Ariadne-transducer** is a walk on the input word, decorated with **states** (from a finite set), that can visit at most k times each position of the input string:



A **transition** is decided based on the full content of the configuration at the head position (input letter + states), and can possibly either

- **[push]** extend the walk of one step (left or right), or
- **[pop]** rewind the walk of one step,

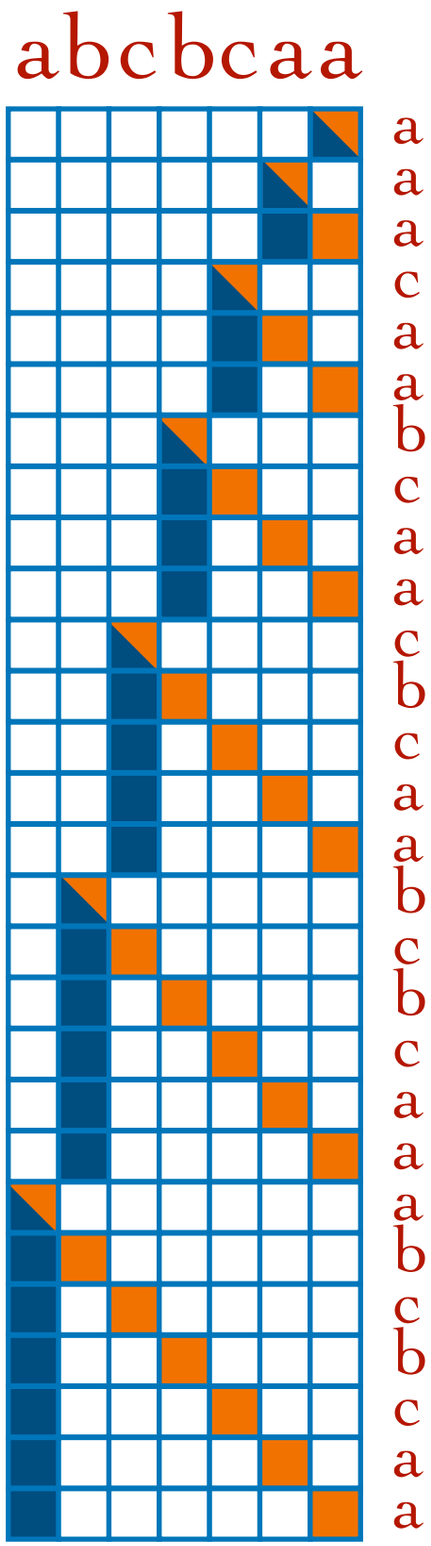
and at the same time it

- **[updates]** the head state, and
- **[outputs]** possibly produces a symbol from the output alphabet.

Also: A **total order** on the states guarantees that it does not loop:

- transitions are required that states increase a each update.

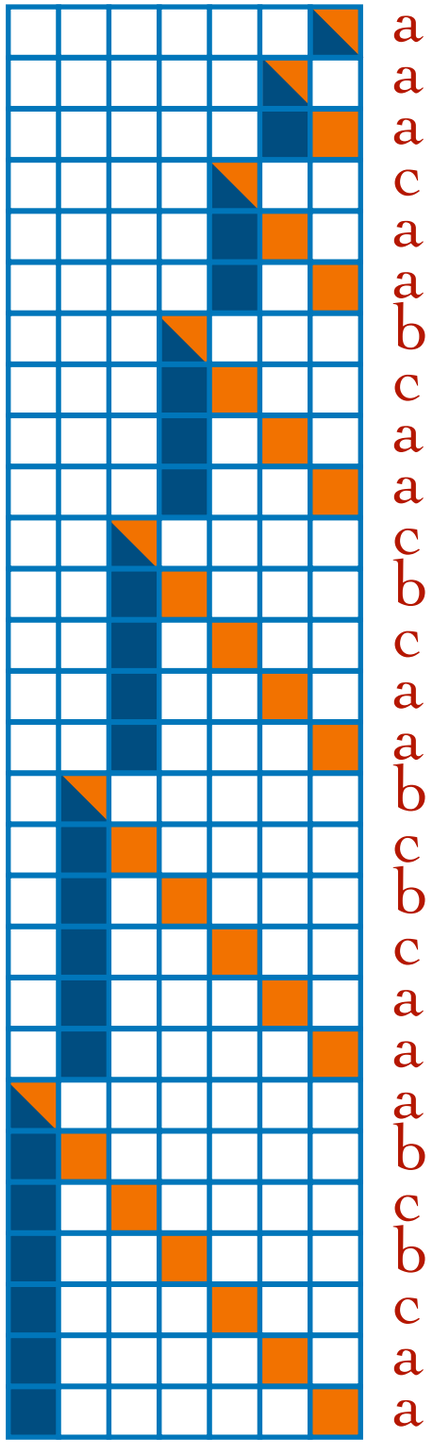
Example (polyregular)



$$f\colon \quad \Sigma^* \quad \to \quad \Sigma^*$$

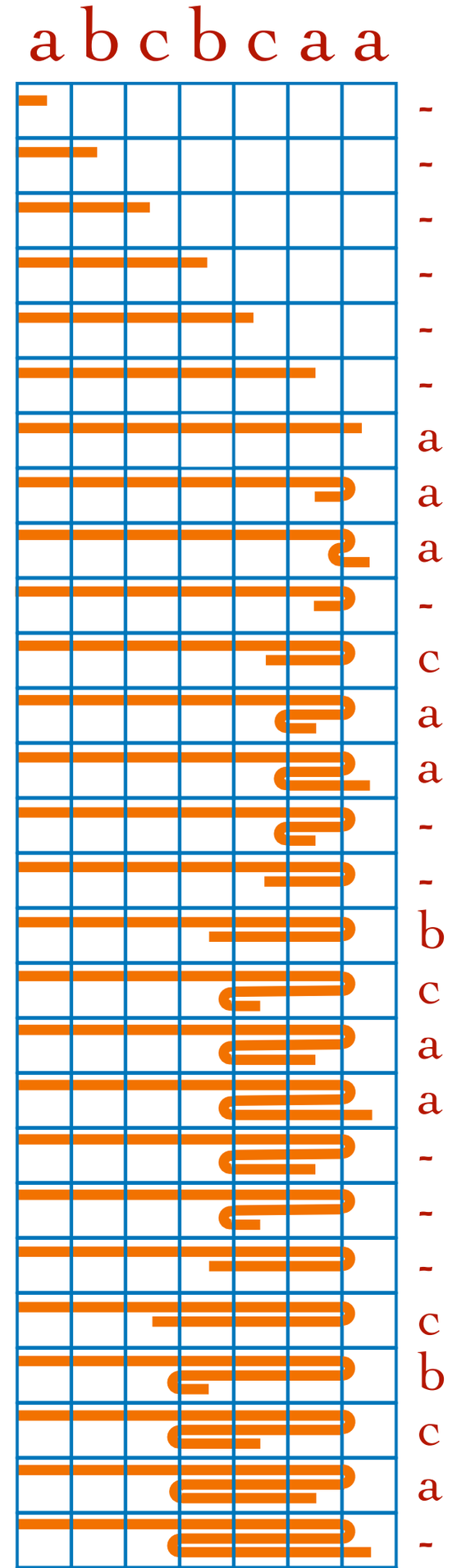
$$a_0 a_1 \dots a_{n-1} \mapsto (a_{n-1})(a_{n-2} a_{n-1}) \dots (a_0 \dots a_{n-1})$$

Example (polyregular)

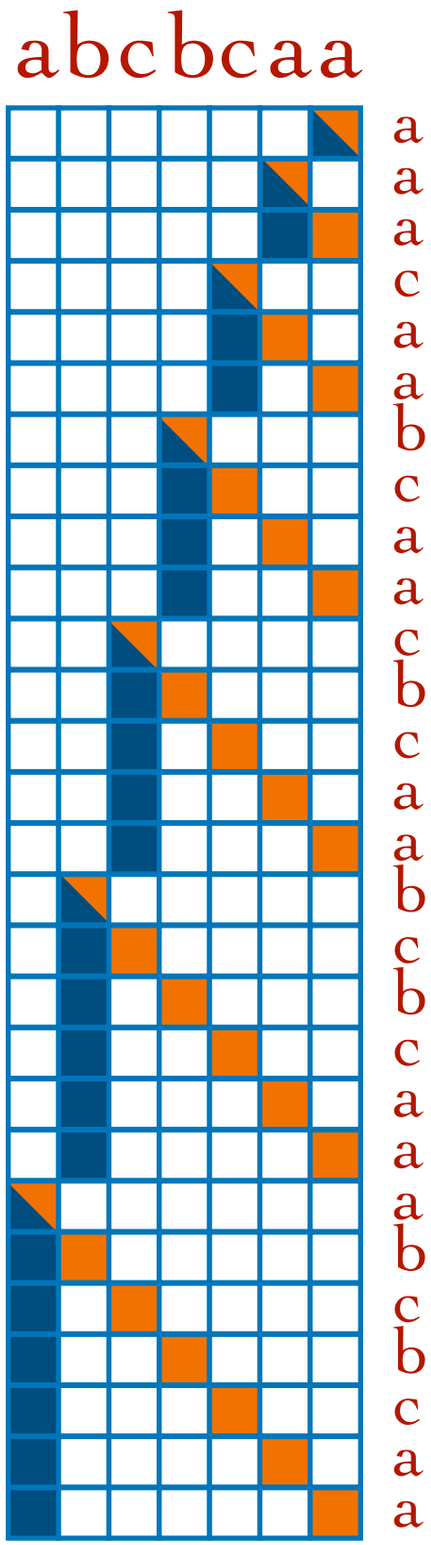


$$f\colon \quad \Sigma^* \quad \to \quad \Sigma^*$$

$$a_0 a_1 \dots a_{n-1} \mapsto (a_{n-1})(a_{n-2} a_{n-1}) \dots (a_0 \dots a_{n-1})$$

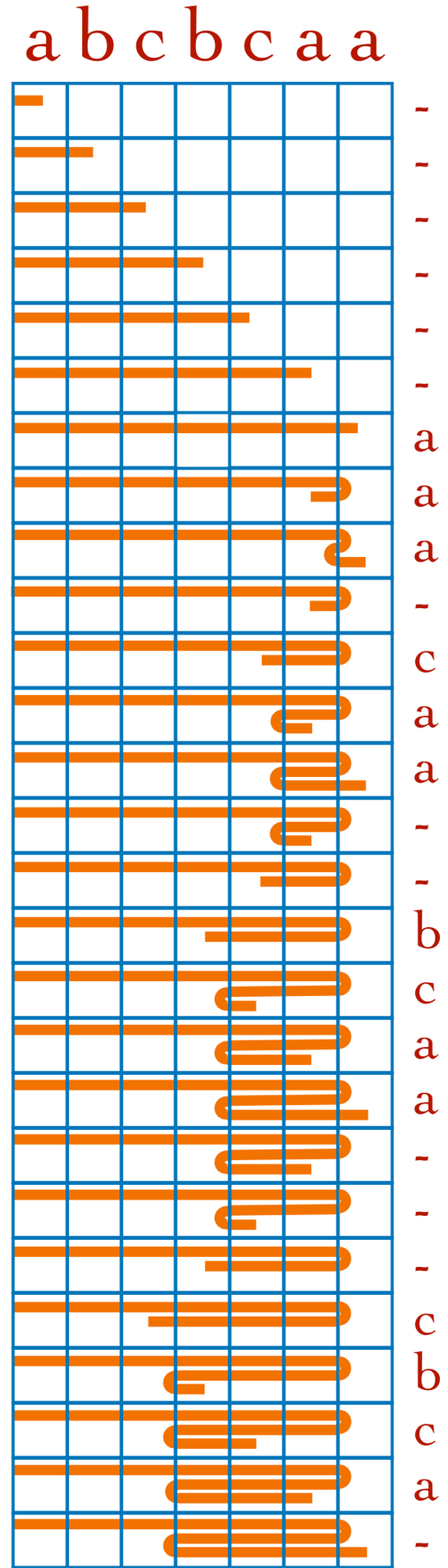


Example (polyregular)



$$f: \quad \Sigma^* \quad \to \quad \Sigma^*$$

$$a_0 a_1 \dots a_{n-1} \mapsto (a_{n-1})(a_{n-2} a_{n-1}) \dots (a_0 \dots a_{n-1})$$

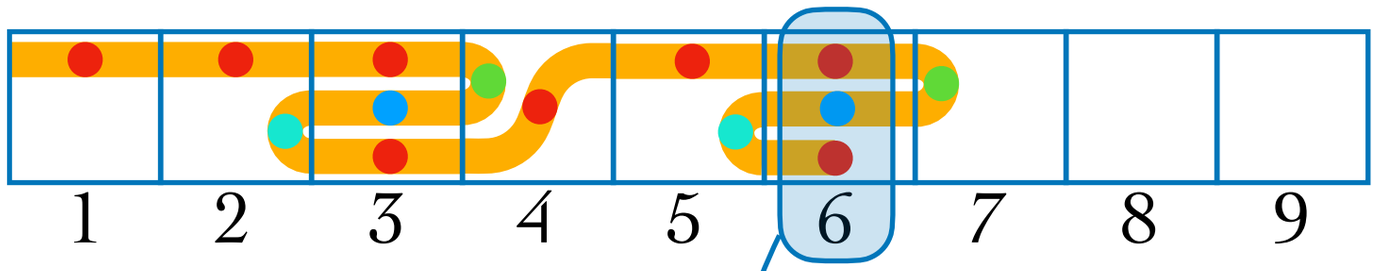


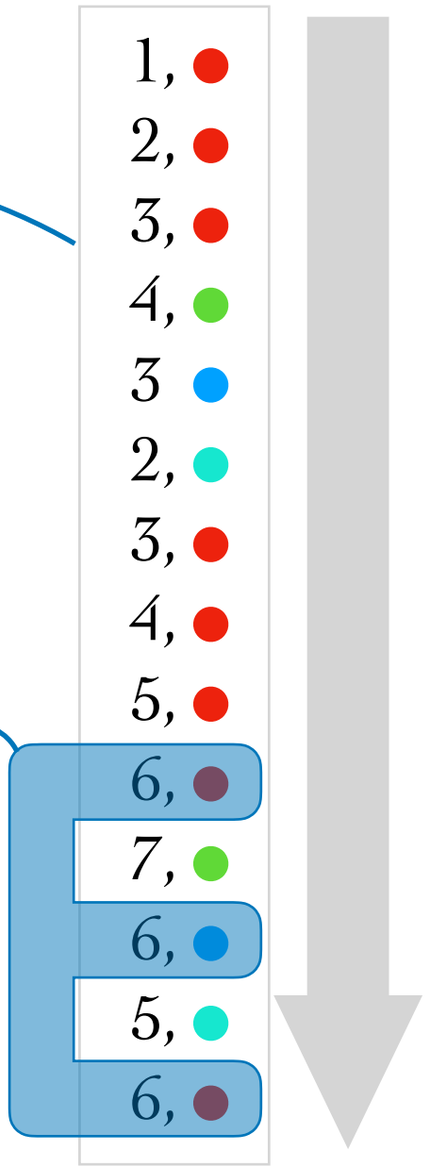
Remark: for polyregular functions, each walk need only use a bounded number of changes of direction.

From Ariadne-transducers to MSO

(Easy direction)

A configuration is a stack of pairs (called global stack):
(position,state)



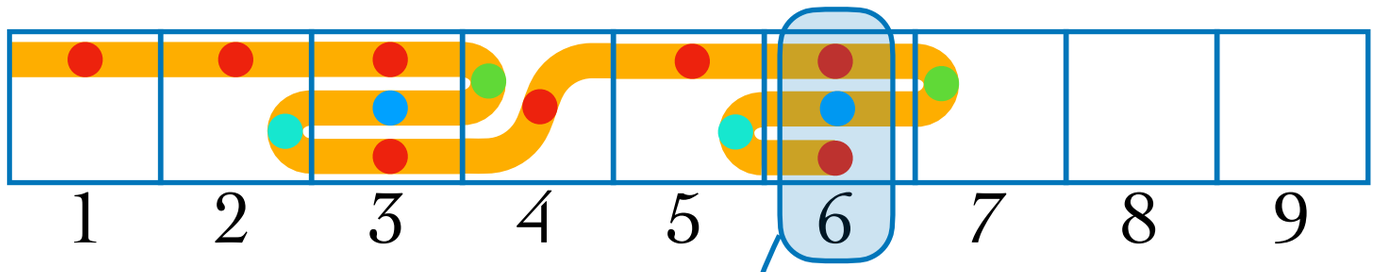


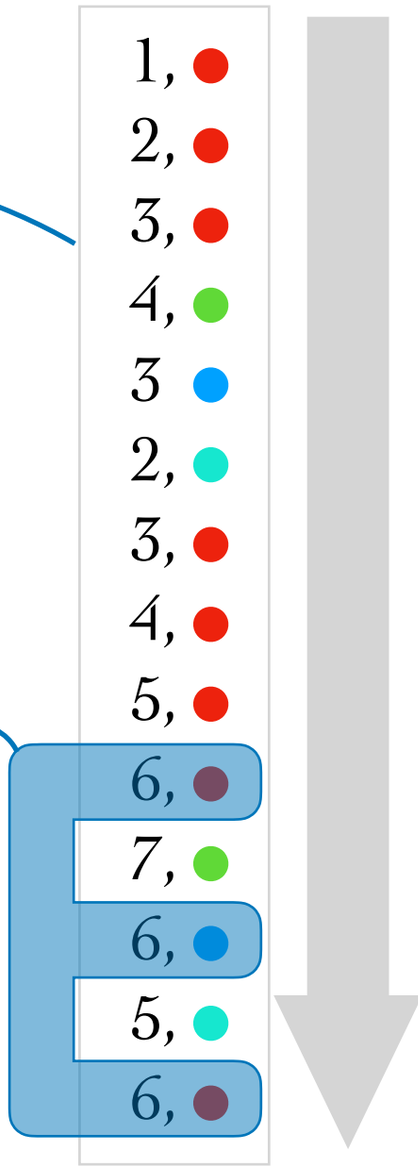
The local stack is its restriction to the head position.

From Ariadne-transducers to MSO

(Easy direction)

A configuration is a stack of pairs (called global stack):
(position,state)





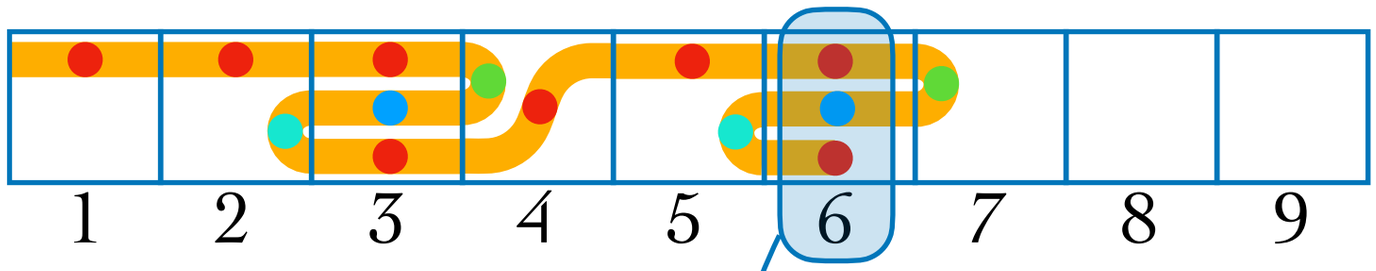
The local stack is its restriction to the head position.

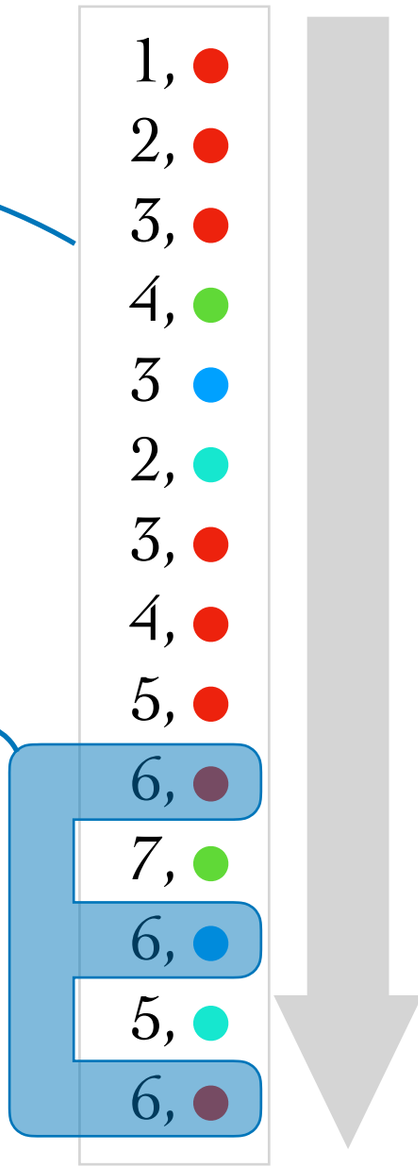
Remark: Global stacks ordered by prefix produce a forest of linear height and bounded rank.

From Ariadne-transducers to MSO

(Easy direction)

A configuration is a stack of pairs (called global stack):
(position,state)





The local stack is its restriction to the head position.

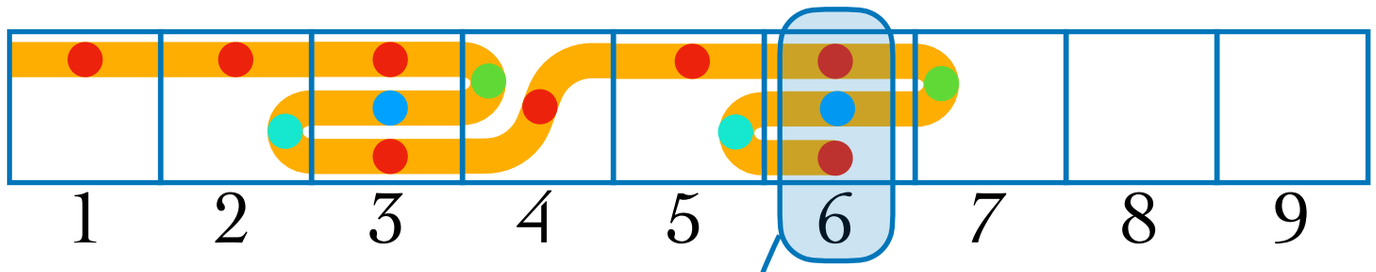
Remark: Global stacks ordered by prefix produce a forest of linear height and bounded rank.

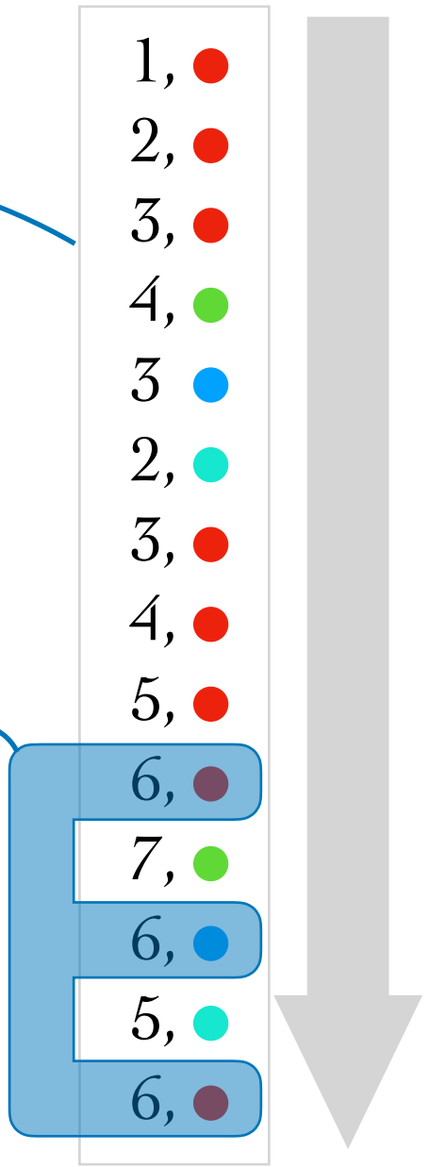
Remark: Configurations can be represented by tuples of sets (=colorings).

From Ariadne-transducers to MSO

(Easy direction)

A configuration is a stack of pairs (called global stack):
(position,state)





The local stack is its restriction to the head position.

Remark: Global stacks ordered by prefix produce a forest of linear height and bounded rank.

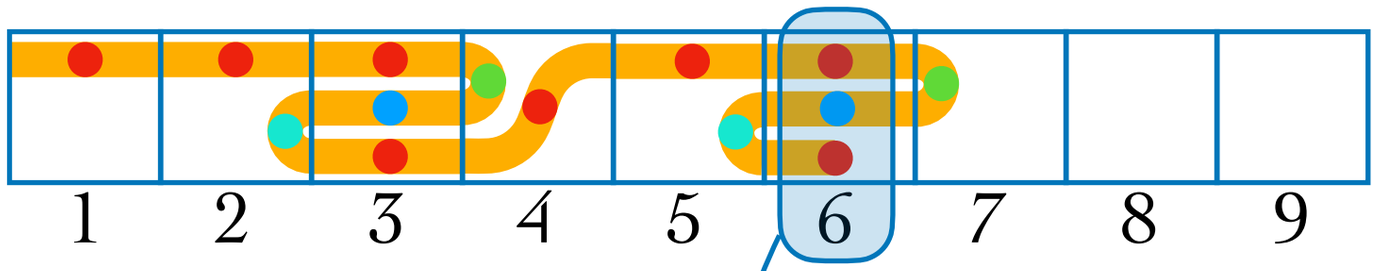
Remark: Configurations can be represented by tuples of sets (=colorings).

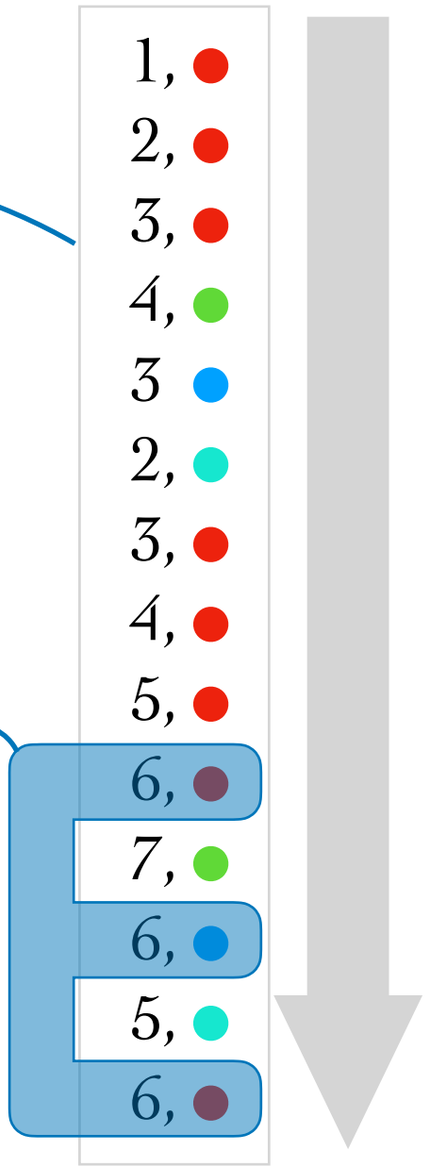
Not obvious: The set of reachable configurations is MSO-definable.

From Ariadne-transducers to MSO

(Easy direction)

A configuration is a stack of pairs (called global stack):
(position,state)





The local stack is its restriction to the head position.

Remark: Global stacks ordered by prefix produce a forest of linear height and bounded rank.

Remark: Configurations can be represented by tuples of sets (=colorings).

Not obvious: The set of reachable configurations is MSO-definable.

Corollary: The output is MSO-set-interpretable in the input. Hence: Functions computed by Ariadne-transducers are MSO-set-interpretations.

Lambda transducers

Consider an input alphabet Σ, and an output alphabet Γ.

A **lambda-transducer** from Σ^* to Γ^*, consists of

- a simple type τ built from the base type o (using $\rightarrow$)
- a simply typed lambda-term $\mathsf{I} : \tau$,
- a simply typed lambda-term $\mathsf{F} : \tau \rightarrow \mathsf{o}$,
- a simply typed lambda-term $\mathsf{T_a} : \tau \rightarrow \tau$, for all a $\in \Sigma$.

Lambda transducers

Consider an input alphabet Σ, and an output alphabet Γ.

A **lambda-transducer** from Σ^* to Γ^*, consists of

- a simple type τ built from the base type o (using $\to$)
- a simply typed lambda-term $\mathsf{I} : \tau$,
- a simply typed lambda-term $\mathsf{F} : \tau \to \mathsf{o}$,
- a simply typed lambda-term $\mathsf{T_a} : \tau \to \tau$, for all a $\in \Sigma$.

Using $\varepsilon : \mathsf{o}$ and
$\mathsf{b} : \mathsf{o} \to \mathsf{o}$, for all b $\in \Gamma$.

Lambda transducers

Consider an input alphabet Σ, and an output alphabet Γ.

A **lambda-transducer** from Σ^* to Γ^*, consists of

- a simple type τ built from the base type o (using $\rightarrow$)
- a simply typed lambda-term $\mathsf{I} : \tau$,
- a simply typed lambda-term $\mathsf{F} : \tau \rightarrow \mathsf{o}$,
- a simply typed lambda-term $\mathsf{T_a} : \tau \rightarrow \tau$, for all $a \in \Sigma$.

Using $\varepsilon : o$ and $b : o \rightarrow o$, for all $b \in \Gamma$.

Given a word $u = a_1 a_2 \ldots a_n \in \Sigma^*$, the lambda-term

$$(T_{a1} ; T_{a2} ; \ldots ; T_{an} ; F)\ I : o$$

Can be totally beta-reduced (in finitely many steps) into a term which has the form (is equivalent to):

$$(b_1 ; b_2 ; \ldots ; b_m)\ \varepsilon$$

Lambda transducers

Consider an input alphabet Σ, and an output alphabet Γ.

A **lambda-transducer** from Σ^* to Γ^*, consists of

- a simple type τ built from the base type o (using $\rightarrow$)
- a simply typed lambda-term $\mathbf{I} : \tau$,
- a simply typed lambda-term $\mathbf{F} : \tau \rightarrow \mathbf{o}$,
- a simply typed lambda-term $\mathbf{T_a} : \tau \rightarrow \tau$, for all $a \in \Sigma$.

Using $\varepsilon : o$ and $b : o \rightarrow o$, for all $b \in \Gamma$.

Given a word $u = a_1 a_2 \ldots a_n \in \Sigma^*$, the lambda-term

$$(T_{a1} ; T_{a2} ; \ldots ; T_{an} ; F)\ I : o$$

; is the reverse composition

Can be totally beta-reduced (in finitely many steps) into a term which has the form (is equivalent to):

$$(b_1 ; b_2 ; \ldots ; b_m)\ \varepsilon$$

Lambda transducers

Consider an input alphabet Σ, and an output alphabet Γ.

A **lambda-transducer** from Σ^* to Γ^*, consists of

- a simple type τ built from the base type o (using $\rightarrow$)
- a simply typed lambda-term $\mathbf{I} : \tau$,
- a simply typed lambda-term $\mathbf{F} : \tau \rightarrow \mathbf{o}$,
- a simply typed lambda-term $\mathbf{T_a} : \tau \rightarrow \tau$, for all a $\in \Sigma$.

Using ε : o and
b : o $\rightarrow$ o, for all b $\in \Gamma$.

Given a word u = $a_1a_2\ldots a_n \in \Sigma^*$, the lambda-term

$$(T_{a1} ; T_{a2} ;\ldots ; T_{an} ; F)\ I : o$$

; is the reverse composition

Can be totally beta-reduced (in finitely many steps) into a term which has the form (is equivalent to):

$$(b_1 ; b_2 ; \ldots ; b_m)\ \varepsilon$$

The **image** of u by the lambda-transducer is $b_1b_2\ldots b_{m-1}b_m$.

Lambda transducers

Example:

- $\tau := o$
- $I := \varepsilon$
- $F := \lambda x.\ x$
- $T_a := a\ ;\ a$
- $T_b := b\ ;\ b$

Lambda transducers

Example:

- $\tau := o$
- $I := \varepsilon$
- $F := \lambda x.\ x$
- $T_a := a\ ;\ a$
- $T_b := b\ ;\ b$

$$(T_{a1}\ ;\ T_{a2}\ ;\ldots\ ;\ T_{an}\ ;\ F)\ I$$

$$=_\beta (a_1\ ;\ a_1\ ;\ a_2\ ;\ a_2\ ;\ \ldots\ ;\ a_n\ ;\ a_n)\ \varepsilon$$

Lambda transducers

Example:

- $\tau := o$
- $I := \varepsilon$
- $F := \lambda x.\ x$
- $T_a := a\ ;\ a$
- $T_b := b\ ;\ b$

$(T_{a1}\ ;\ T_{a2}\ ;\ldots\ ;\ T_{an}\ ;\ F)\ I$

$=_\beta (a_1\ ;\ a_1\ ;\ a_2\ ;\ a_2\ ;\ \ldots\ ;\ a_n\ ;\ a_n)\ \varepsilon$

Copies the input, while doubling each letter.

Lambda transducers

Example:

- $\tau := o$
- $I := \varepsilon$
- $F := \lambda x.\ x$
- $T_a := a\ ;\ a$
- $T_b := b\ ;\ b$

$$(T_{a1}\ ;\ T_{a2}\ ;\ldots\ ;\ T_{an}\ ;\ F)\ I$$
$$=_\beta (a_1\ ;\ a_1\ ;\ a_2\ ;\ a_2\ ;\ \ldots\ ;\ a_n\ ;\ a_n)\ \varepsilon$$

Copies the input, while doubling each letter.

Example:

- $\tau := o \longrightarrow o$
- $I := \lambda x.\ x$
- $F := \lambda f.\ f\ \varepsilon$
- $T_a := \lambda f.\ a\ ;\ f$
- $T_b := \lambda f.\ b\ ;\ f$

Lambda transducers

Example:

- $\tau := o$
- $I := \varepsilon$
- $F := \lambda x.\ x$
- $T_a := a\ ;\ a$
- $T_b := b\ ;\ b$

$$(T_{a1}\ ;\ T_{a2}\ ;\ \ldots\ ;\ T_{an}\ ;\ F)\ I$$
$$=_\beta (a_1\ ;\ a_1\ ;\ a_2\ ;\ a_2\ ;\ \ldots\ ;\ a_n\ ;\ a_n)\ \varepsilon$$

Copies the input, while doubling each letter.

Example:

- $\tau := o \longrightarrow o$
- $I := \lambda x.\ x$
- $F := \lambda f.\ f\ \varepsilon$
- $T_a := \lambda f.\ a\ ;\ f$
- $T_b := \lambda f.\ b\ ;\ f$

Mirrors the input

Lambda transducers

Example:

- $\tau := o$
- $I := \varepsilon$
- $F := \lambda x.\ x$
- $T_a := a\ ;\ a$
- $T_b := b\ ;\ b$

$$(T_{a1}\ ;\ T_{a2}\ ;\ldots\ ;\ T_{an}\ ;\ F)\ I$$
$$=_\beta (a_1\ ;\ a_1\ ;\ a_2\ ;\ a_2\ ;\ \ldots\ ;\ a_n\ ;\ a_n)\ \varepsilon$$

Copies the input, while doubling each letter.

Example:

- $\tau := o \longrightarrow o$
- $I := \lambda x.\ x$
- $F := \lambda f.\ f\ \varepsilon$
- $T_a := \lambda f.\ a\ ;\ f$
- $T_b := \lambda f.\ b\ ;\ f$

Mirrors the input

Lemma: Ariadne-transducers can be effectively transformed into equivalent lambda-transducers.

Lambda transducers

Example:

- $\tau := o$
- $I := \varepsilon$
- $F := \lambda x.\ x$
- $T_a := a\ ;\ a$
- $T_b := b\ ;\ b$

$$(T_{a1}\ ;\ T_{a2}\ ;\ldots\ ;\ T_{an}\ ;\ F)\ I$$
$$=_\beta (a_1\ ;\ a_1\ ;\ a_2\ ;\ a_2\ ;\ \ldots\ ;\ a_n\ ;\ a_n)\ \varepsilon$$

Copies the input, while doubling each letter.

Example:

- $\tau := o \longrightarrow o$
- $I := \lambda x.\ x$
- $F := \lambda f.\ f\ \varepsilon$
- $T_a := \lambda f.\ a\ ;\ f$
- $T_b := \lambda f.\ b\ ;\ f$

Mirrors the input

Lemma: Ariadne-transducers can be effectively transformed into equivalent lambda-transducers.

The parameter k corresponds to the order of τ.

Inverse images of regular sets

Lemma: The inverse image of a regular language by a lambda-transducers is regular.

Inverse images of regular sets

Lemma: The inverse image of a regular language by a lambda-transducers is regular.

Proof: Consider a DFA for a language L, of states Q, and interpret

- the type o as the state Q
- $\tau \longrightarrow \tau'$ as the maps from the interpretation of τ to the interpretations of τ'
- ε as the initial state (in o)
- each letter a as its action on Q.

Inverse images of regular sets

Lemma: The inverse image of a regular language by a lambda-transducers is regular.

Proof: Consider a DFA for a language L, of states Q, and interpret

- the type o as the state Q
- $\tau \longrightarrow \tau'$ as the maps from the interpretation of τ to the interpretations of τ'
- ε as the initial state (in o)
- each letter a as its action on Q.

Then the interpretation of the lambda-transducer is a DFA recognising the inverse image of L by the lambda transducer.

Inverse images of regular sets

Lemma: The inverse image of a regular language by a lambda-transducers is regular.

Proof: Consider a DFA for a language L, of states Q, and interpret

- the type o as the state Q
- $\tau \longrightarrow \tau'$ as the maps from the interpretation of τ to the interpretations of τ'
- ε as the initial state (in o)
- each letter a as its action on Q.

Then the interpretation of the lambda-transducer is a DFA recognising the inverse image of L by the lambda transducer.

Corollary: MSO-set-interpretations preserve regular language of strings by inverse image.

Conclusion

Finite state transducers

Interpretations

Finite model theory

Simply typed
lambda-calculus

Conclusion

Finite state transducers

Interpretations

Finite model theory

Simply typed
lambda-calculus

These correspondances form an active topic of research that tie together two sides of logic: model theory and proof theory!

Conclusion

Finite state transducers

Interpretations

Finite model theory

Conjecture: Reverse holds under an exponential growth assumption.

Simply typed lambda-calculus

These correspondances form an active topic of research that tie together two sides of logic: model theory and proof theory!