Decomposition of finite-valued streaming string transducers

Paul Gallot
Anca Muscholl
Gabriele Puppis
Sylvain Salvati

Transformations of objects, here words

transduction = function or relation between words

Transformations of objects, here words

transduction = function or relation between words

hannover $\longrightarrow$	{hannover}*	Kleene iteration
hannover $\longrightarrow$	revonnah	mirror
hannover $\longrightarrow$	hannover hannover	duplicate
hannover $\longrightarrow$	over hann	split & swap

MSOT = monadic second-order transductions [Courcelle '95]

Logically define the output inside copies of the input:

- domain: unary formula selecting positions in each copy
- order: binary formula defining an order on the domain
- letters: unary formulas partitioning the domain

hannover ⟶ revonnah mirror

$\varphi_<(x,y)$ = "$x > y$" // $x < y$ in the output iff $x > y$ in the input

Transductions defined by automata

Finite-state Transducers = automata with outputs on transitions

	Deterministic	Non-deterministic
One-way	**1DFT**	**1NFT**
Two-way	**2DFT**	**2NFT**

SST = Streaming String Transducers

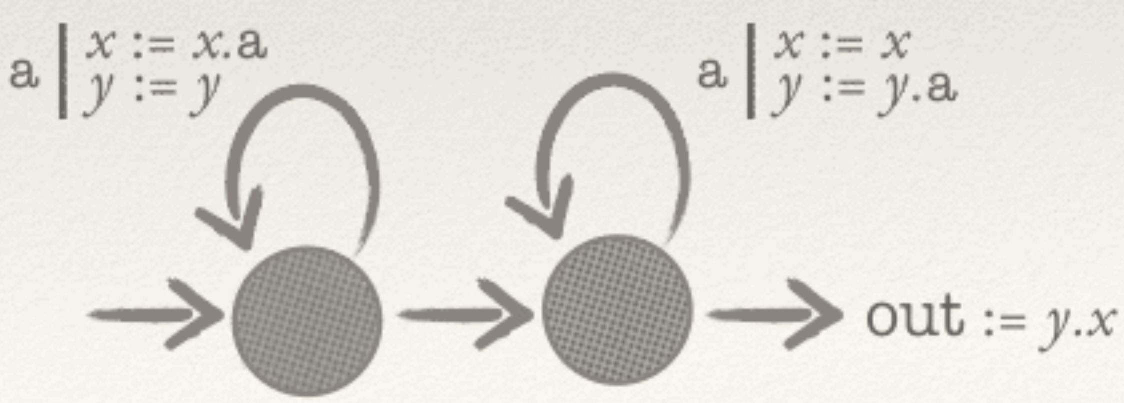
[Alur, Cerny '10]

- deterministic / non-deterministic
- 1-way
- write-only registers to store partial outputs
  + copyless restriction = each register used at most once

$x =$

$y =$

E.g. split & swap



SST = Streaming String Transducers

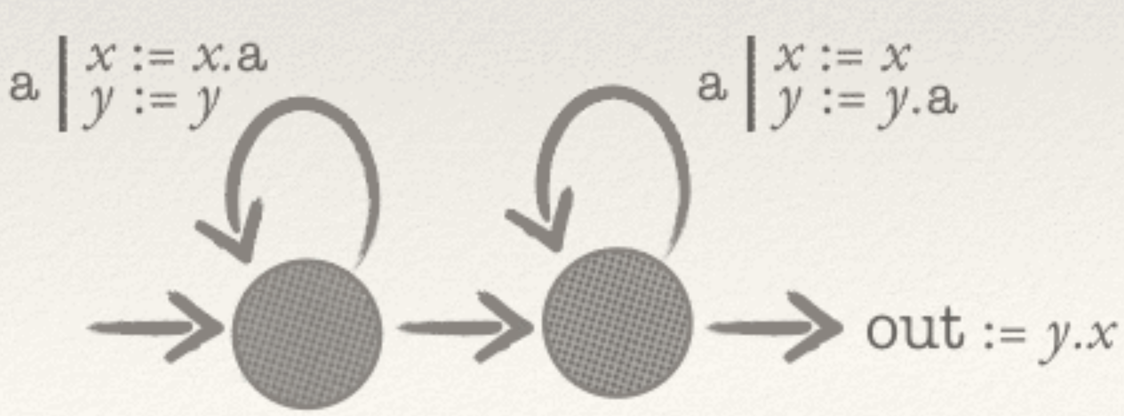
[Alur, Cerny '10]

- deterministic / non-deterministic
- 1-way
- write-only registers to store partial outputs
  \+ copyless restriction = each register used at most once

$x =$

$y =$

E.g. split & swap



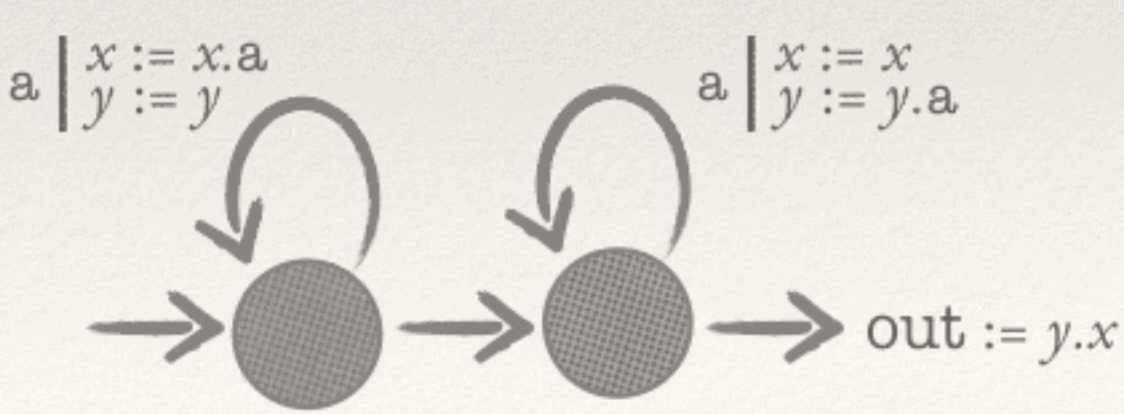
SST = Streaming String Transducers [Alur, Cerny '10]

- deterministic / non-deterministic
- 1-way
- write-only registers to store partial outputs
  + copyless restriction = each register used at most once

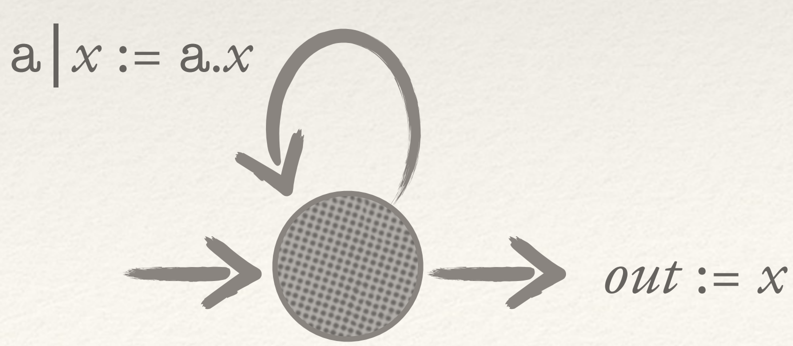
$x =$

$y =$

E.g. split & swap

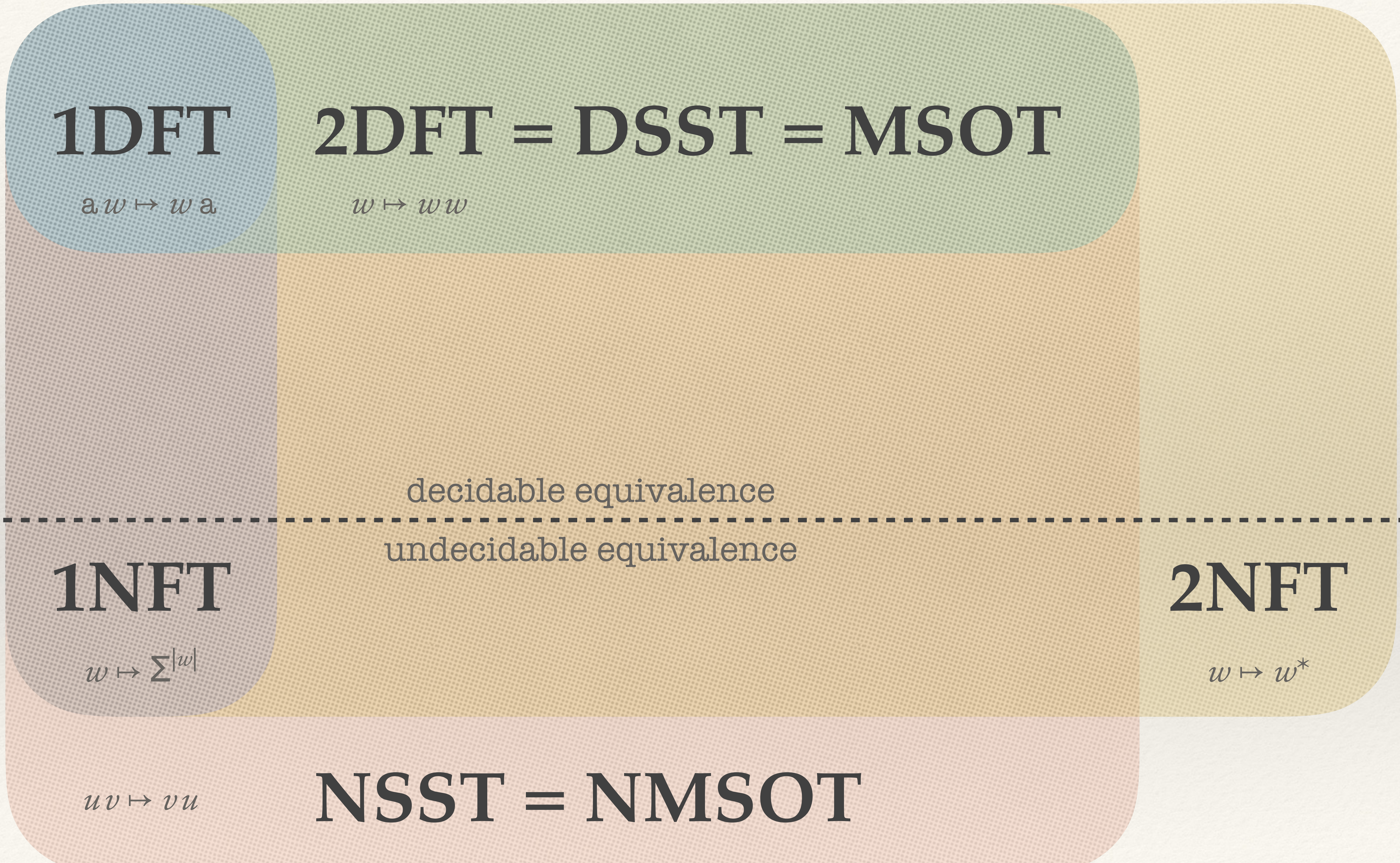


E.g. mirror



Relational transductions

Relational transductions



Functional
~~Relational~~ transductions

1DFT **2DFT = DSST = MSOT**

$\mathrm{a}\,w \mapsto w\,\mathrm{a}$ $w \mapsto w\,w$

||

1NFT **NSST = NMSOT = 2NFT**

$w\,\mathrm{a} \mapsto \mathrm{a}\,w$

decidable equivalence

undecidable equivalence

Functional

~~Relational~~ transductions

1DFT

$\mathrm{a}\,w \mapsto w\,\mathrm{a}$

2DFT = DSST = MSOT

$w \mapsto w\,w$

||

1NFT

$w\,\mathrm{a} \mapsto \mathrm{a}\,w$

NSST = NMSOT = 2NFT

decidable equivalence

---

undecidable equivalence

Anything interesting beyond functional transductions?

k-valued transductions = at most *k* outputs for each input

- decidable equivalence?
- correspondence with logic (e.g. MSO) ?
- equivalent models (e.g. 2-way vs SSTs) ?
- effective characterisations (e.g. 1-way definability) ?

k-valued transductions = at most k outputs for each input

- decidable equivalence?
- correspondence with logic (e.g. MSO) ?
- equivalent models (e.g. 2-way vs SSTs) ?
- effective characterisations (e.g. 1-way definability) ?

a unifying approach:

Decomposition Theorem

For a suitable class $\mathscr{C}$ of transducers:

"Every k-valued transducer $\in \mathscr{C}$ can be decomposed into a finite union of functional transducers $\in \mathscr{C}$ "

The decomposition theorem for $\mathscr{C}$ = { 1NFTs } :

$\boxed{\text{Every } k\text{-valued 1NFT is a finite union of functional 1NFTs.}}$

[Weber '96, Sakarovitch - de Souza '08]

The decomposition theorem for $\mathscr{C}$ = { 1NFTs } :

$$\boxed{\text{Every } k\text{-valued 1NFT is a finite union of functional 1NFTs.}}$$

[Weber ’96, Sakarovitch - de Souza ‘08]

Corollaries:

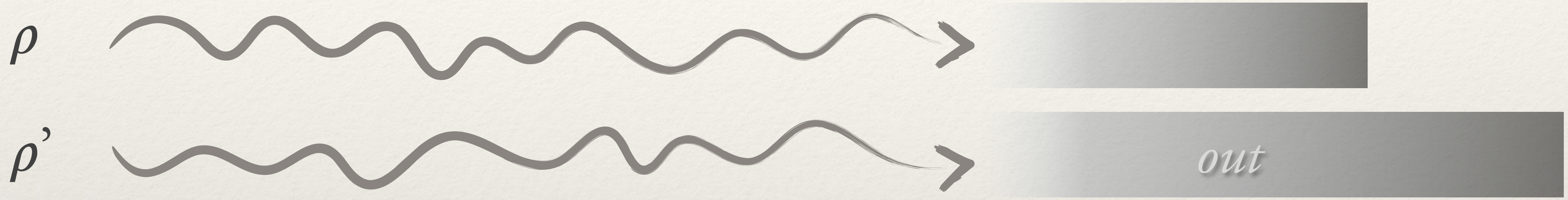
- decidable equivalence of k-valued 1NFTs
- k-valued 1NFTs = k-valued order-preserving MSO transductions

Unboundedly many runs with same output…

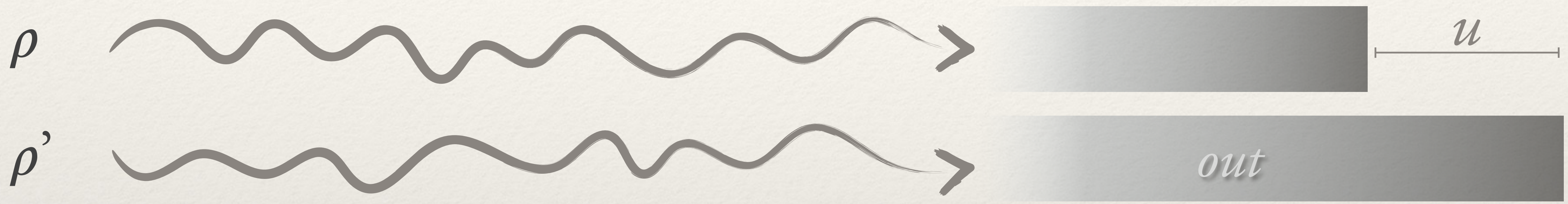
Follow lexico.-least run for each output

Subproblem: compare runs by their outputs

In a 1NFT, outputs are formed by appending only to the right

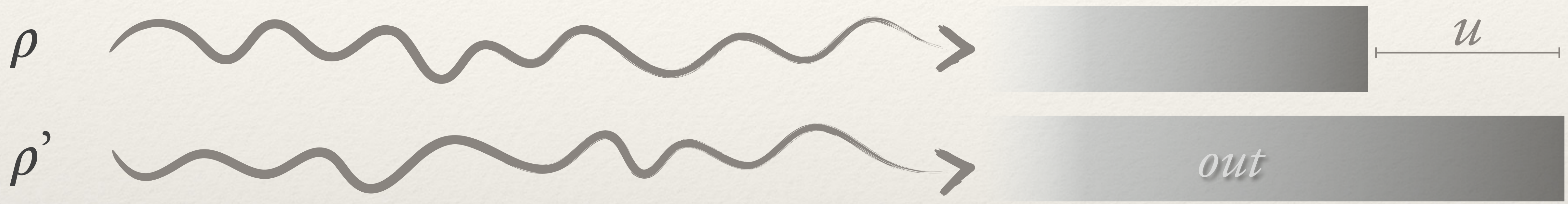


In a 1NFT, outputs are formed by appending only to the right



$$\textcolor{blue}{align\,(\rho,\rho')} = \begin{cases} (u, \varepsilon) \;:\; out\,(\rho)\,.\,u = out\,(\rho') \\ (\varepsilon, v) \;:\; out\,(\rho) = out\,(\rho')\,.\,v \end{cases}$$

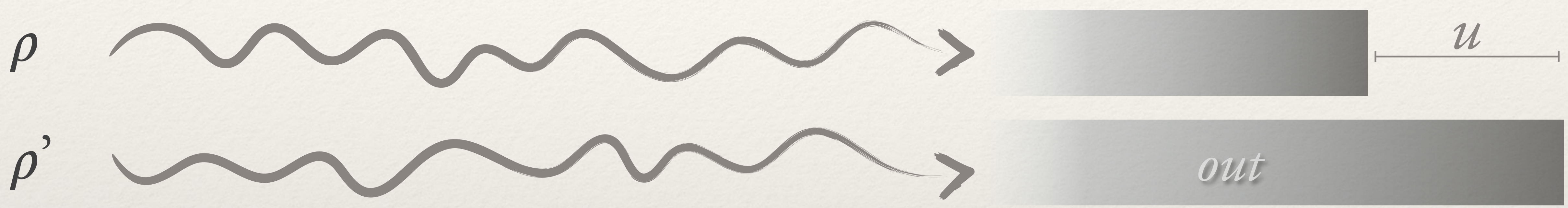
In a 1NFT, outputs are formed by appending only to the right



$$align(\rho,\rho') = \begin{cases} (u, \varepsilon) & : \; out(\rho) \,.\, u = out(\rho') \\ (\varepsilon, v) & : \; out(\rho) = out(\rho') \,.\, v \end{cases}$$

$$lag(\rho,\rho') = \left|\, align(\rho,\rho') \,\right|$$

In a 1NFT, outputs are formed by appending only to the right



$$align(\rho,\rho') = \begin{cases} (u, \varepsilon) \; : \; out(\rho)\,.\,u = out(\rho') \\ (\varepsilon, v) \; : \; out(\rho) = out(\rho')\,.\,v \end{cases}$$

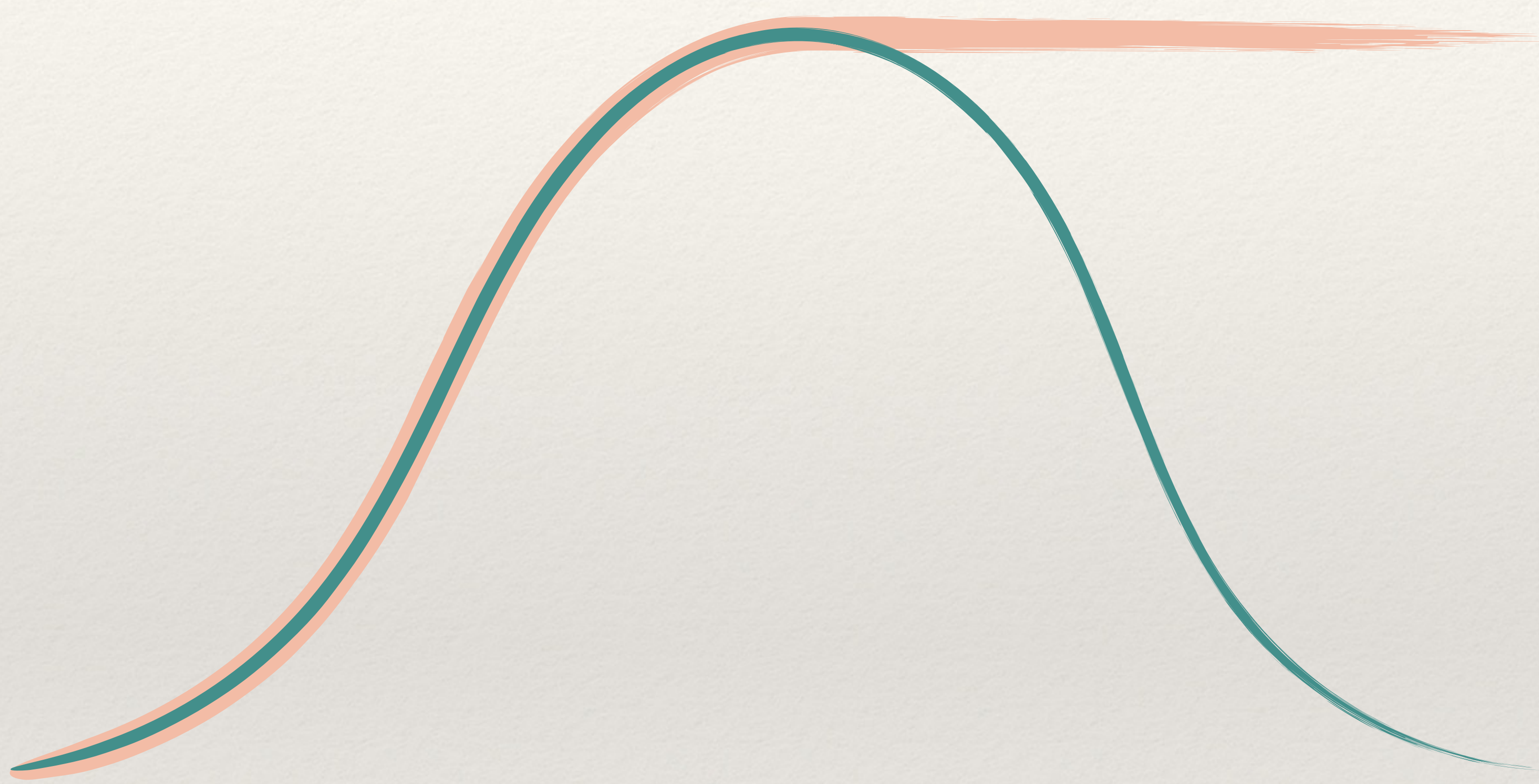
$$lag(\rho,\rho') = |\, align(\rho,\rho')\, |$$

$$maxlag(\rho,\rho') = \mathrm{MAX}\, \{\, lag(\rho_{\leq t}, \rho'_{\leq t}) \; : \; t \leq |\rho| \,\}$$

It may happen that $out(\rho) = out(\rho')$ yet with large $maxlag(\rho,\rho')$

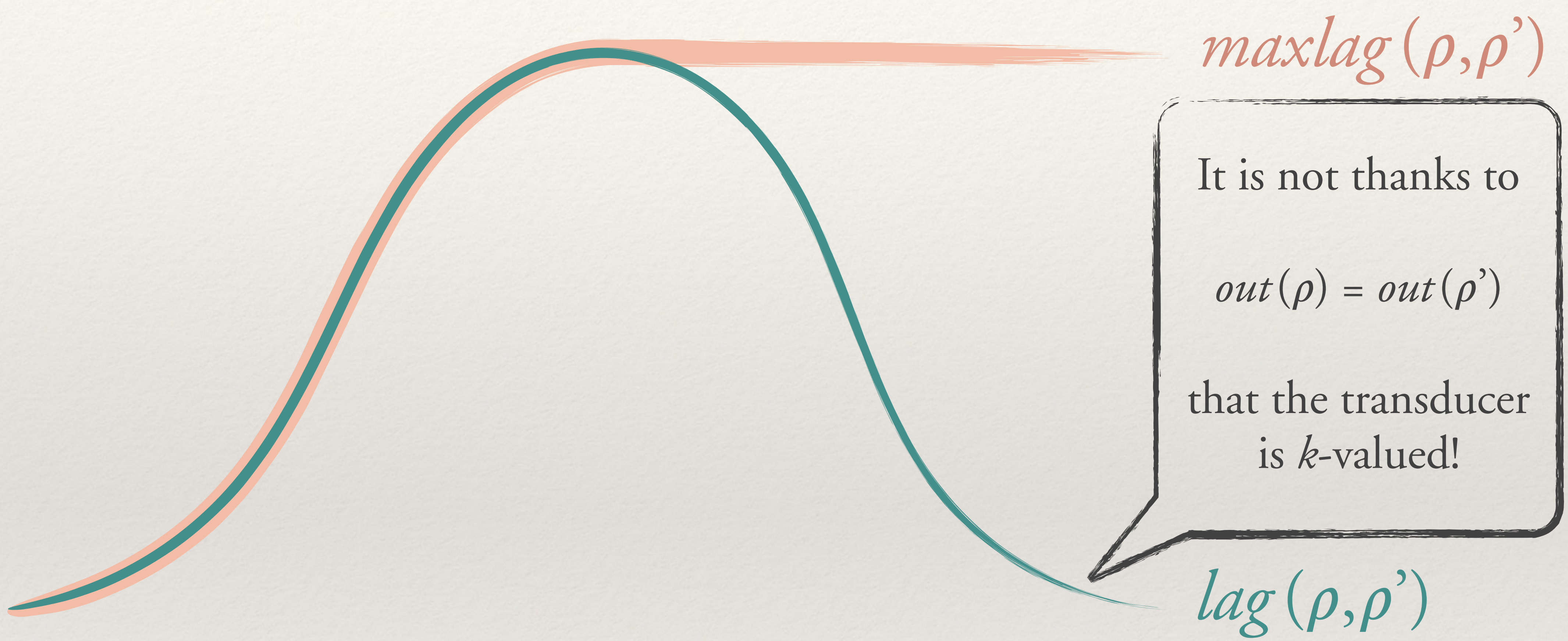
input	=	a a ... a a ...	#	... a a ... a a
$out(\rho)$	=	a a ... a a ...		
$out(\rho')$	=			... a a ... a a

It may happen that $out(\rho) = out(\rho')$ yet with large $maxlag(\rho,\rho')$



input	=	a a ... a a ... # ... a a ... a a
$out(\rho)$	=	a a ... a a ...
$out(\rho')$	=	... a a ... a a

It may happen that $out(\rho) = out(\rho')$ yet with large $maxlag(\rho,\rho')$



input	=	a a … a a …	#	… a a … a a
$out(\rho)$	=	a a … a a …		
$out(\rho')$	=			… a a … a a

Key combinatorial property:

$\rho_1, \ldots, \rho_{k+1}$ runs of k-valued 1NFT $\Rightarrow$ $\exists\, i \neq j$ $\quad out(\rho_i) = out(\rho_j)$ & $maxlag(\rho_i, \rho_j)$ small

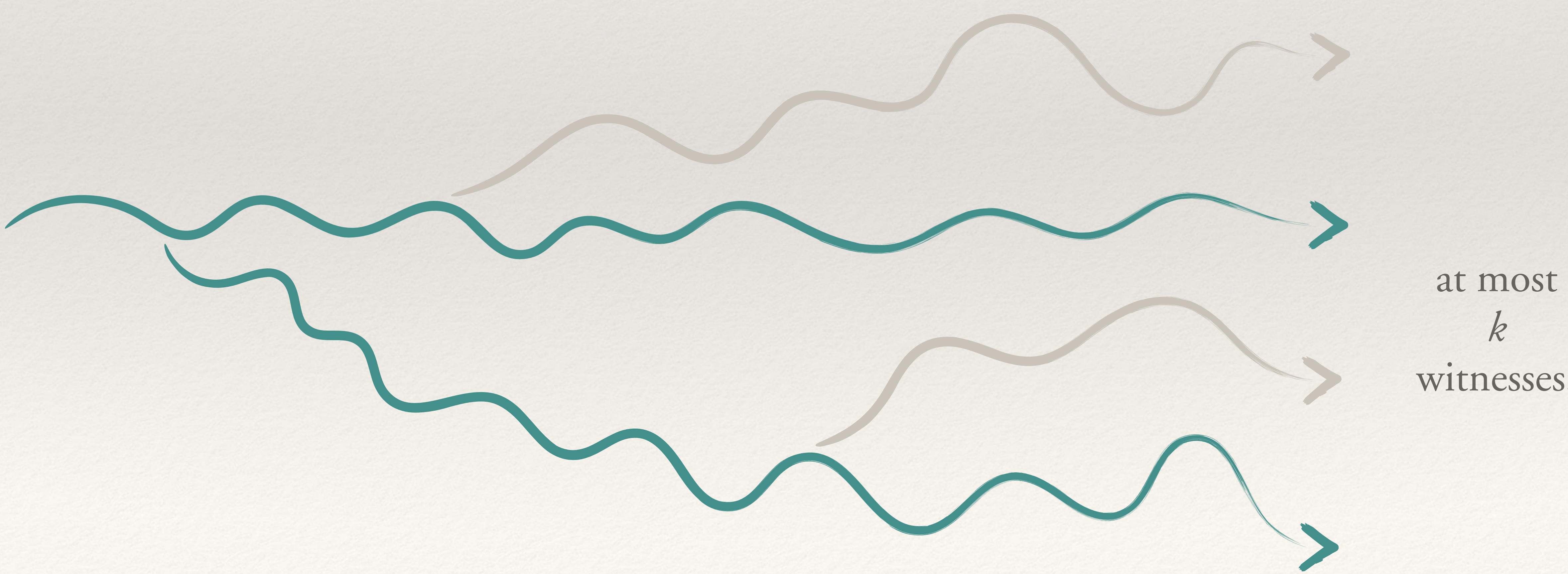
Key combinatorial property:

$\rho_1, \ldots, \rho_{k+1}$ runs of k-valued 1NFT $\Rightarrow$ $\exists\, i \neq j$ $\quad out(\rho_i) = out(\rho_j)$ & $maxlag(\rho_i, \rho_j)$ small

Moreover, if $maxlag(\rho_i, \rho_j)$ is small one can maintain $align(\rho_i, \rho_j)$ in bounded memory — in particular, one knows whether $out(\rho_i) = out(\rho_j)$

One can simulate only the **witness runs**, namely, the ρ's that are

- successful
- lexico.-least among all other runs ρ' with $\begin{cases} out(\rho) = out(\rho') \ \& \\ maxlag(\rho, \rho') \text{ small} \end{cases}$



$\boxed{\text{Conjecture: every } k\text{-valued SST is a finite union of functional SSTs}}$

Conjecture: every k-valued SST is a finite union of functional SSTs

Some corollaries:

- decidable equivalence of k-valued SSTs [Alur, Deshmukh '11]
- k-valued SSTs = k-valued MSO transductions [Alur, Cerny '10]
  = k-valued 2NFTs [Engelfriet, Hoogeboom '01]
- effective characterisation of k-valued SSTs definable by k-valued 1NFTs [Filiot, Gauwin, Reynier, Servais '13]

Conjecture: every k-valued SST is a finite union of functional SSTs

Some corollaries:

$T \subseteq T_1 \cup \ldots \cup T_k$ decidable
for functional SSTs $T, T_1, \ldots, T_k$

- decidable equivalence of k-valued SSTs [Alur, Deshmukh '11]
- k-valued SSTs = k-valued MSO transductions [Alur, Cerny '10]
  = k-valued 2NFTs [Engelfriet, Hoogeboom '01]
- effective characterisation of k-valued SSTs definable by k-valued 1NFTs [Filiot, Gauwin, Reynier, Servais '13]

Conjecture: every *k*-valued SST is a finite union of functional SSTs

Some corollaries:

in the functional case

DSSTs = MSO = 2NFTs

- decidable equivalence of *k*-valued SSTs [Alur, Deshmukh '11]
- *k*-valued SSTs = *k*-valued MSO transductions [Alur, Cerny '10]
  = *k*-valued 2NFTs [Engelfriet, Hoogeboom '01]
- effective characterisation of *k*-valued SSTs definable by *k*-valued 1NFTs [Filiot, Gauwin, Reynier, Servais '13]

Conjecture: every k-valued SST is a finite union of functional SSTs

Some corollaries:

- decidable equivalence of k-valued SSTs [Alur, Deshmukh '11]

- k-valued SSTs = k-valued MSO transductions [Alur, Cerny '10]
  = k-valued 2NFTs [Engelfriet, Hoogeboom '01]

- effective characterisation of k-valued SSTs definable by k-valued 1NFTs [Filiot, Gauwin, Reynier, Servais '13]

Our contribution: we proved the conjecture for SSTs with 1 register

First difficulty: letters added to left and right of register $\Rightarrow$ symmetric alignments on registers

$$align(\rho,\rho') = \{ \lambda=(u, v, w, z) \; : \; u \,.\, reg(\rho) \,.\, v = w \,.\, reg(\rho') \,.\, z \}$$

First difficulty: letters added to left and right of register $\Rightarrow$ symmetric alignments on registers

$$align(\rho,\rho') = \{ \lambda=(u, v, w, z) \;:\; u\,.\,\mathbf{reg}(\rho)\,.\,v = w\,.\,\mathbf{reg}(\rho')\,.\,z \}$$

$$lag(\rho,\rho') = \mathrm{MIN}\,\{ |\lambda| \;:\; \lambda \in align(\rho,\rho') \}$$

First difficulty: letters added to left and right of register $\Rightarrow$ symmetric alignments on registers

$$align(\rho,\rho') = \{\ \lambda=(u,v,w,z)\ :\ u\,.\,\mathbf{reg}(\rho)\,.\,v = w\,.\,\mathbf{reg}(\rho')\,.\,z\ \}$$

$$lag(\rho,\rho') = \mathrm{MIN}\ \{\ |\lambda|\ :\ \lambda \in align(\rho,\rho')\ \}$$

$$maxlag(\rho,\rho') = \mathrm{MAX}\ \{\ lag(\rho_{\leq t}, \rho'_{\leq t})\ :\ t \leq |\rho|\ \}$$

Decomposition of 1-register SSTs

Second difficulty: combinatorial property

$$\boxed{\begin{array}{lcl} k{+}1 \text{ runs } \overline{\rho} = \rho_1, \ldots, \rho_{k+1} & & \exists\, i \neq j \quad reg(\rho_i) = reg(\rho_j) \ \& \\ \text{of } k\text{-valued SST} & \Rightarrow & maxlag(\rho_i, \rho_j) \text{ small} \end{array}}$$

Some equalities hold just by chance...

Second difficulty: combinatorial property

$(i,j) \in \textbf{Equals}(\overline{\rho})$

k+1 runs $\overline{\rho} = \rho_1, \ldots, \rho_{k+1}$ of k-valued SST $\Rightarrow$ $\exists\, i \neq j$ $\quad reg(\rho_i) = reg(\rho_j)$ & $maxlag(\rho_i, \rho_j)$ small

$(i,j) \in \textbf{SmallLags}(\overline{\rho})$

Some equalities hold just by chance...

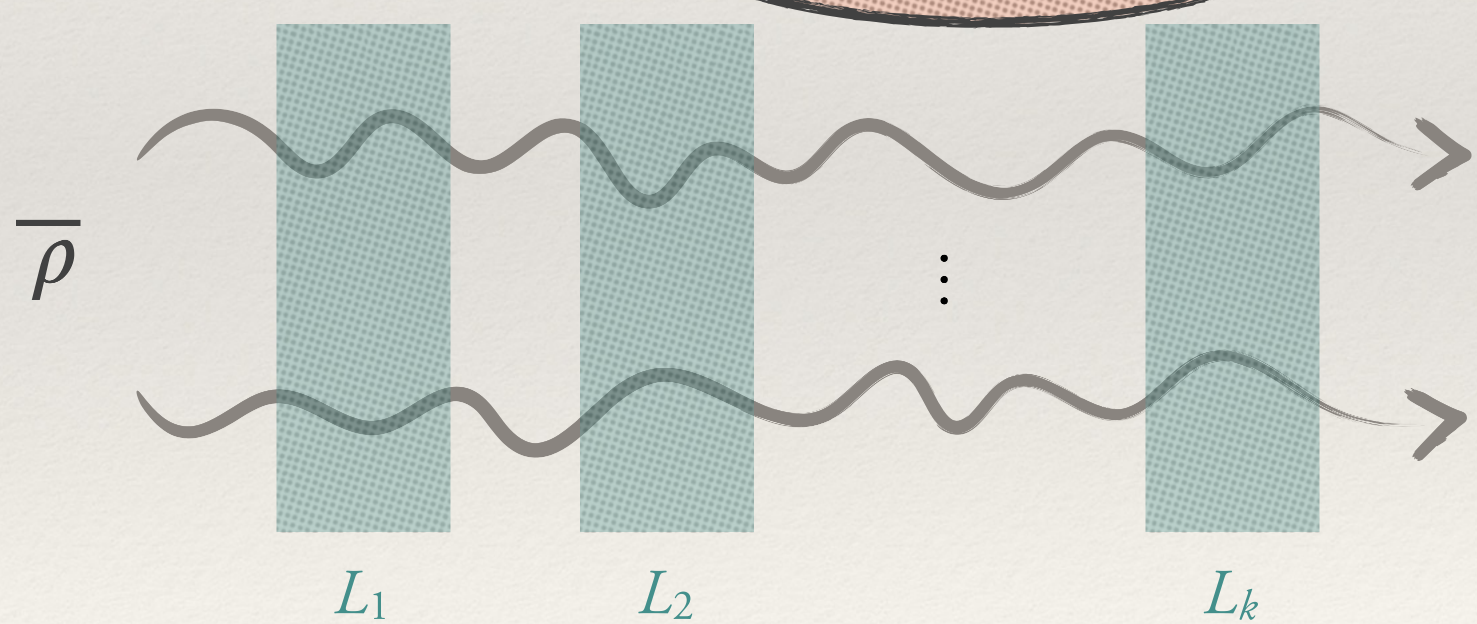
Second difficulty: combinatorial property

$(i,j) \in \mathbf{Equals}(\overline{\rho})$

k+1 runs $\overline{\rho} = \rho_1, \ldots, \rho_{k+1}$ of k-valued SST $\Rightarrow$	$\exists\, i \neq j$ $\quad reg(\rho_i) = reg(\rho_j)$ & $maxlag(\rho_i, \rho_j)$ small

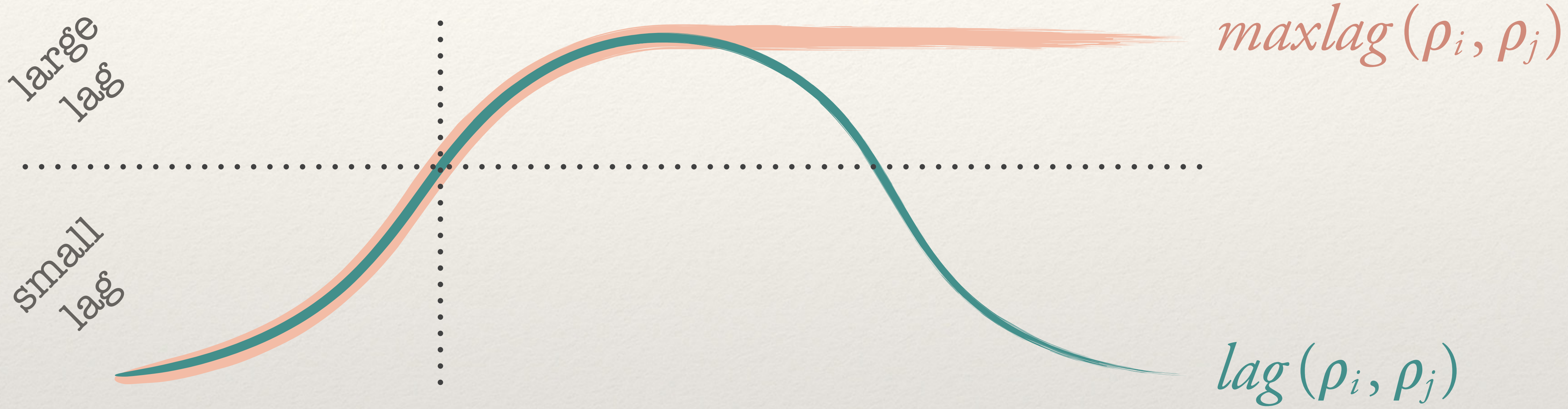
$(i,j) \in \mathbf{SmallLags}(\overline{\rho})$

Some equalities hold just by chance...

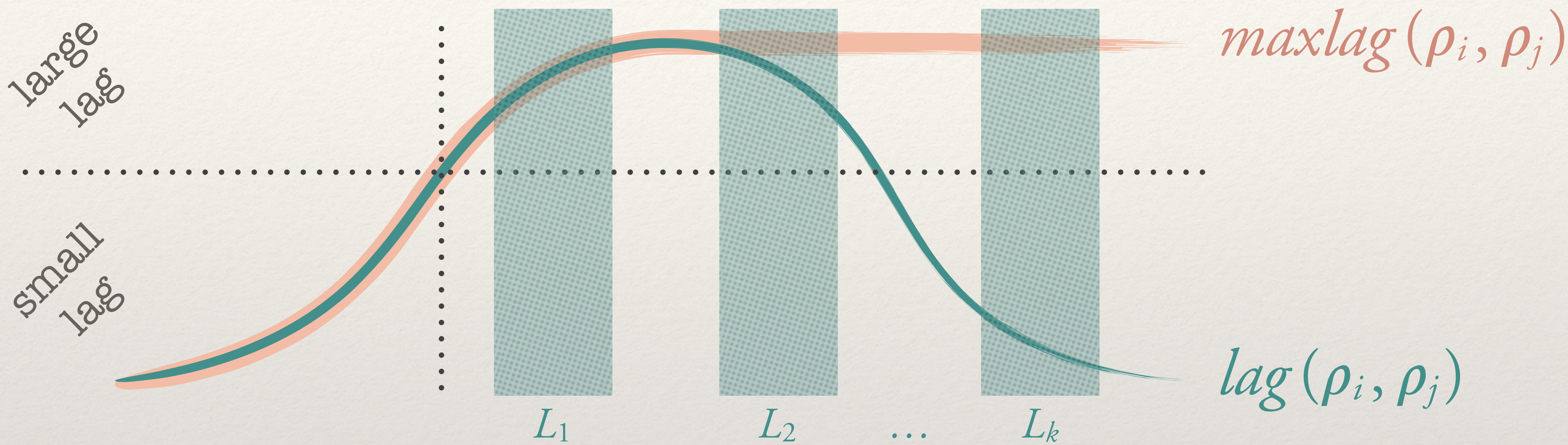


$(i,j) \in \mathbf{Equals}(\overline{\rho})$ is not robust to pumping

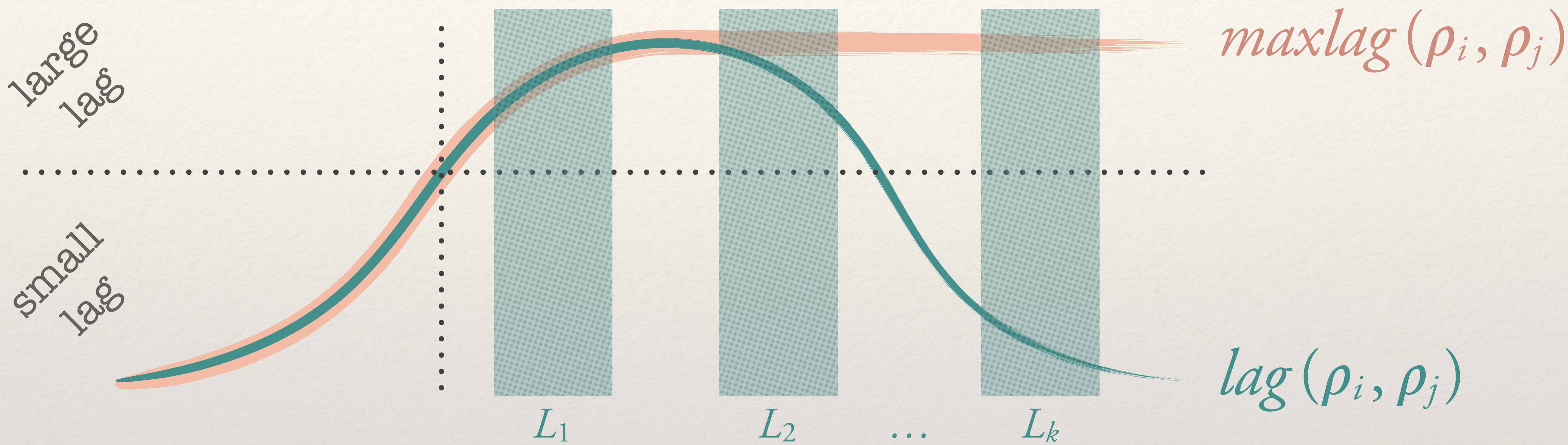
Take $(i,j) \in$ **Equals**$(\overline{\rho})$



Take $(i,j) \in$ **Equals**$(\overline{\rho})$ Let $\overline{\rho}_n = pump^n_{L_1,\ldots,L_k}(\overline{\rho})$



Take $(i,j) \in$ **Equals**$(\overline{\rho})$ Let $\overline{\rho}_n = pump^n_{L_1,\ldots,L_k}(\overline{\rho})$

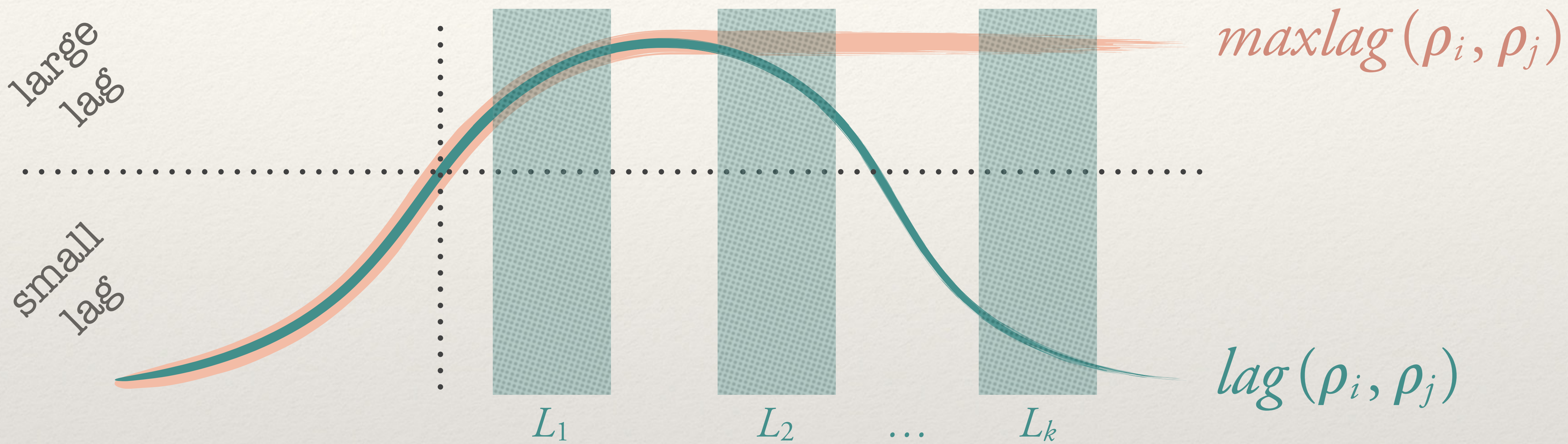


2 cases

$(i,j) \in$ **Equals**$(\overline{\rho}_n)$ holds for **only finitely many** n's

$(i,j) \in$ **Equals**$(\overline{\rho}_n)$ holds for **infinitely many** n's

Take $(i,j) \in$ **Equals**$(\overline{\rho})$ Let $\overline{\rho}_n = pump^n_{L_1,\dots,L_k}(\overline{\rho})$



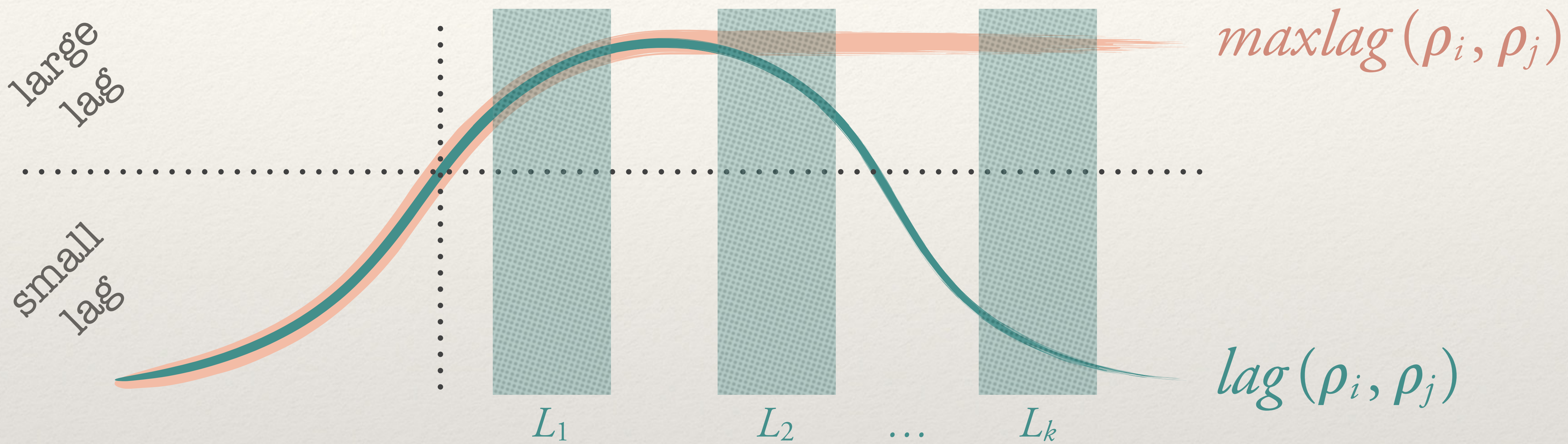
2 cases

$(i,j) \in$ **Equals**$(\overline{\rho}_n)$ holds for **only finitely many** n**'s**

$\Rightarrow$ replace $\overline{\rho}$ by $\overline{\rho}_n$ for large enough n

$(i,j) \in$ **Equals**$(\overline{\rho}_n)$ holds for **infinitely many** n**'s**

Take $(i,j) \in \mathbf{Equals}(\overline{\rho})$ Let $\overline{\rho}_n = pump^n_{L_1,\ldots,L_k}(\overline{\rho})$



2 cases

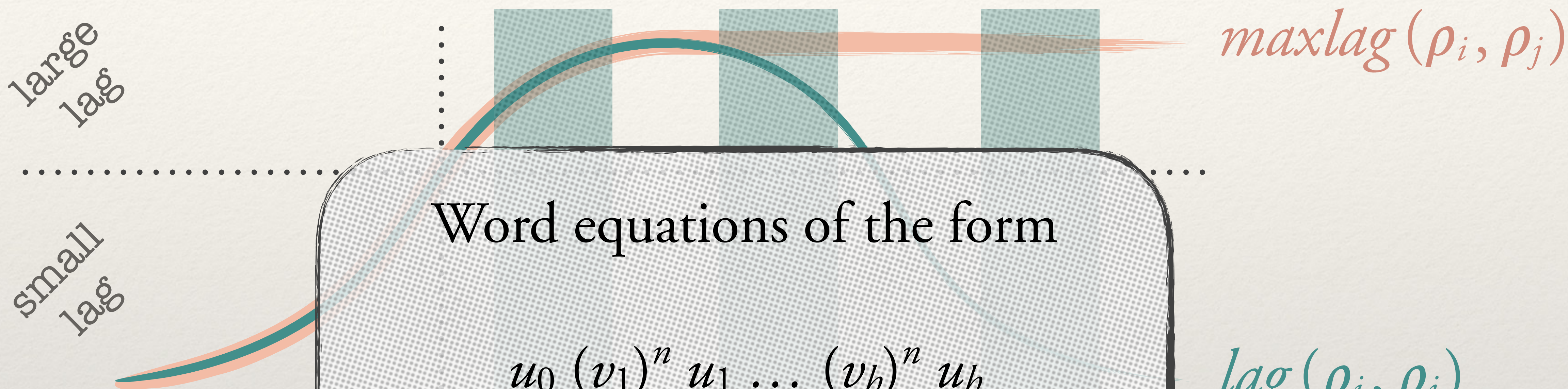
$(i,j) \in \mathbf{Equals}(\overline{\rho}_n)$ holds for **only finitely many** n's

$\Rightarrow$ replace $\overline{\rho}$ by $\overline{\rho}_n$ for large enough n

$(i,j) \in \mathbf{Equals}(\overline{\rho}_n)$ holds for **infinitely many** n's

$\Rightarrow$ it holds **for all** n**'s**, including n=0 …

Take $(i,j) \in$ **Equals**$(\overline{\rho})$ Let $\overline{\rho}_n = pump^n_{L_1,\dots,L_k}(\overline{\rho})$



Word equations of the form

$$u_0\,(v_1)^n\,u_1 \dots (v_h)^n\,u_h = u'_0\,(v'_1)^n\,u'_1 \dots (v'_h)^n\,u'_h$$

2 cases

$(i,j) \in$ **Equ**

$\Rightarrow$ $\overline{\rho}_n$ for large enough n

$(i,j) \in$ **Equals**$(\overline{\rho}_n)$ … **inite**

[Kortelainen '98, Saarela '15] $\Rightarrow$ it holds **for all** n**'s**, including n=0 …

Third difficulty: maintain alignments in bounded memory

Third difficulty: maintain alignments in bounded memory

After updating

$$reg(\rho) := reg(\rho) . \mathtt{s}$$
$$reg(\rho') := \mathtt{a} . reg(\rho')$$

Third difficulty: maintain alignments in bounded memory

After updating

$reg(\rho) := reg(\rho)\,.\,\mathtt{s}$

$reg(\rho') := \mathtt{a}\,.\,reg(\rho')$

+ knowledge on periodicity:

$reg(\rho) \in \{\ \mathtt{stac}\ \}^*$

$reg(\rho') \in \{\ \mathtt{csta}\ \}^*$

Decomposition of 1-register SSTs

Third difficulty: maintain alignments in bounded memory

After updating

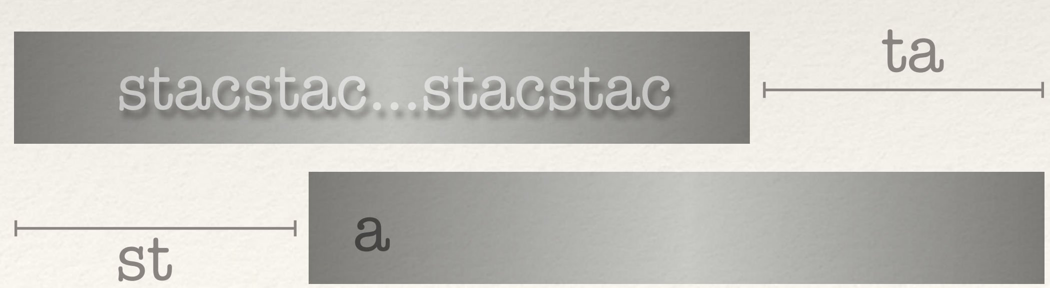
$reg(\rho) := reg(\rho) \,.\, \mathtt{s}$

$reg(\rho') := \mathtt{a} \,.\, reg(\rho')$

+ knowledge on periodicity:

$reg(\rho) \in \{\, \mathtt{stac} \,\}^*$

$reg(\rho') \in \{\, \mathtt{csta} \,\}^*$



Theorem
Every k-valued SST with 1 register is a union of k functional SSTs.

Corollary
Equivalence problem for k-valued SSTs with 1 register is decidable.

A first steps towards a decomposition theorem for SSTs with many registers…

Managed to prove the combinatorial property with many registers:

$\rho_1, \ldots, \rho_{k+1}$ runs of k-valued SST	$\Rightarrow$	$\exists\, i \neq j \quad out(\rho_i) = out(\rho_j)$ & $maxlag(\rho_i, \rho_j)$ small

Managed to prove the combinatorial property with many registers:

$\rho_1, \ldots, \rho_{k+1}$ runs of k-valued SST	$\Rightarrow$	$\exists\, i \neq j \quad out(\rho_i) = out(\rho_j)$ & $maxlag(\rho_i, \rho_j)$ small

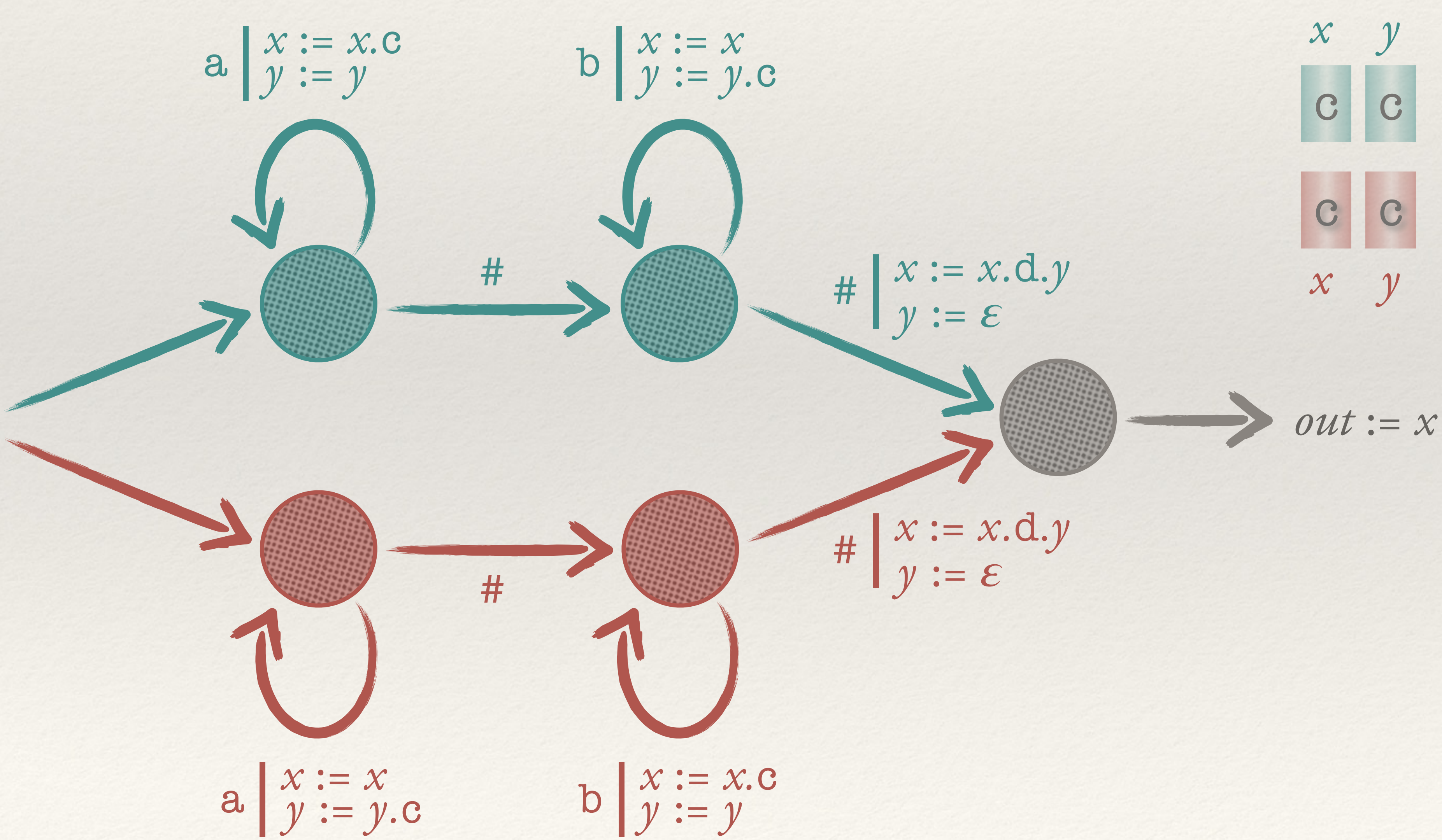
Idea:

1. not all loops induce repetitions of factors in the registers
2. those that do not induce repetitions can be simulated with less registers
3. word equations + induction on number of registers…

$\boxed{\textit{maxlag}(\rho_i, \rho_j) \text{ small} \quad \Rightarrow \quad \textit{align}(\rho_i, \rho_j) \text{ maintainable in bounded memory}}$

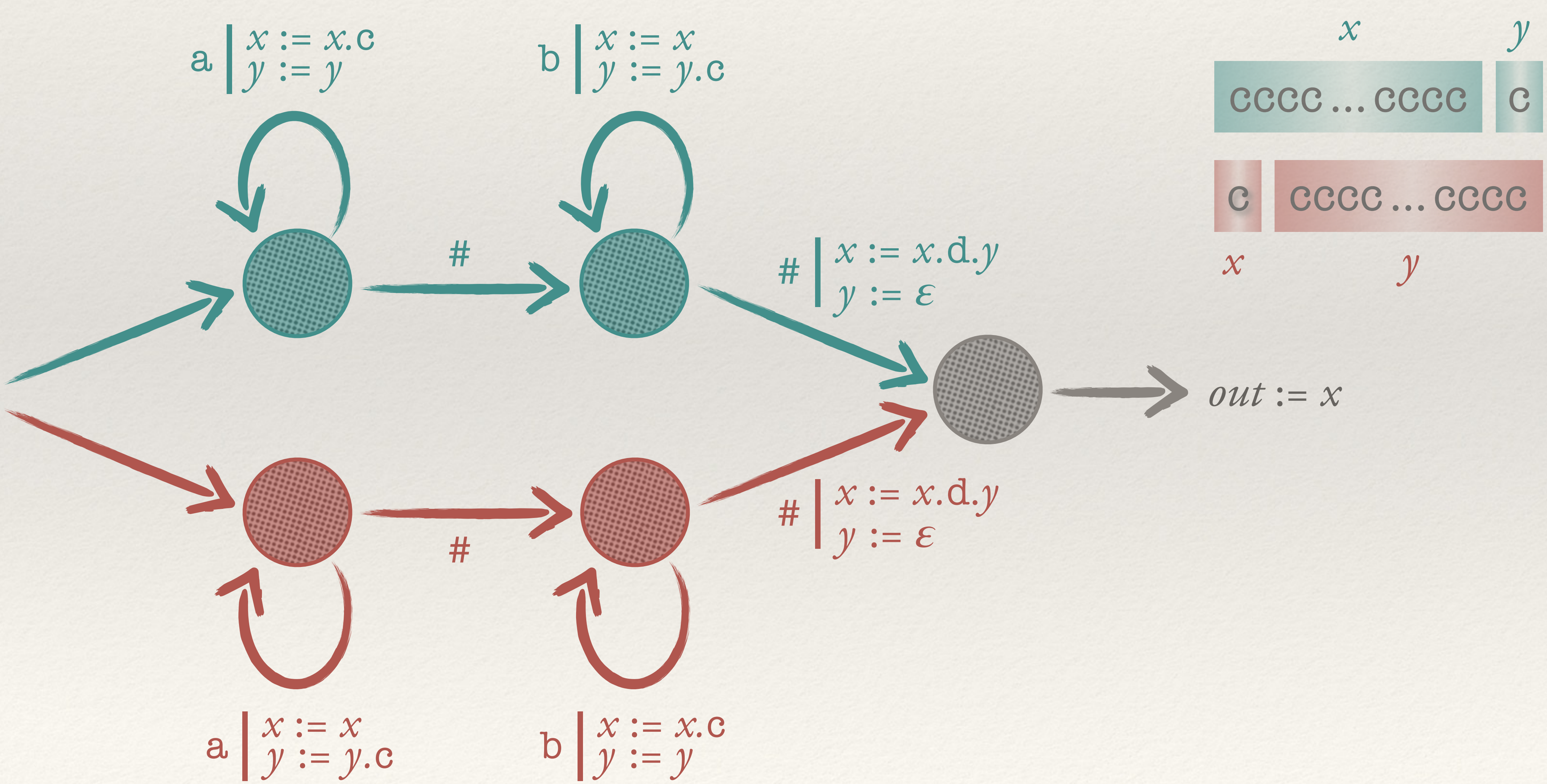
$maxlag(\rho_i, \rho_j)$ small $\Rightarrow$ $align(\rho_i, \rho_j)$ maintainable in bounded memory

FALSE

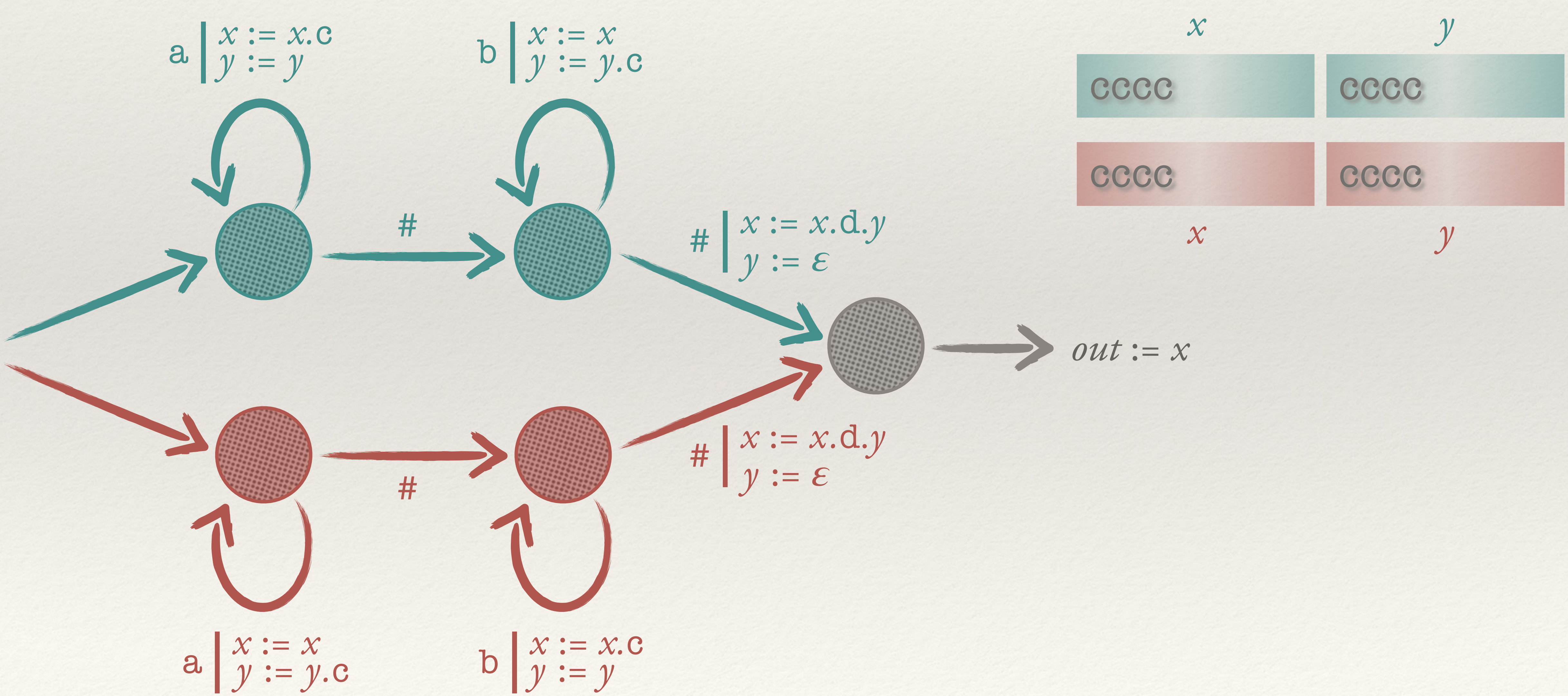


$maxlag(\rho_i, \rho_j)$ small $\Rightarrow$ $align(\rho_i, \rho_j)$ maintainable in bounded memory

FALSE



$maxlag(\rho_i, \rho_j)$ small FALSE $\Rightarrow$ $align(\rho_i, \rho_j)$ maintainable in bounded memory



$maxlag(\rho_i, \rho_j)$ small $\Rightarrow$ $align(\rho_i, \rho_j)$ maintainable in bounded memory

FALSE