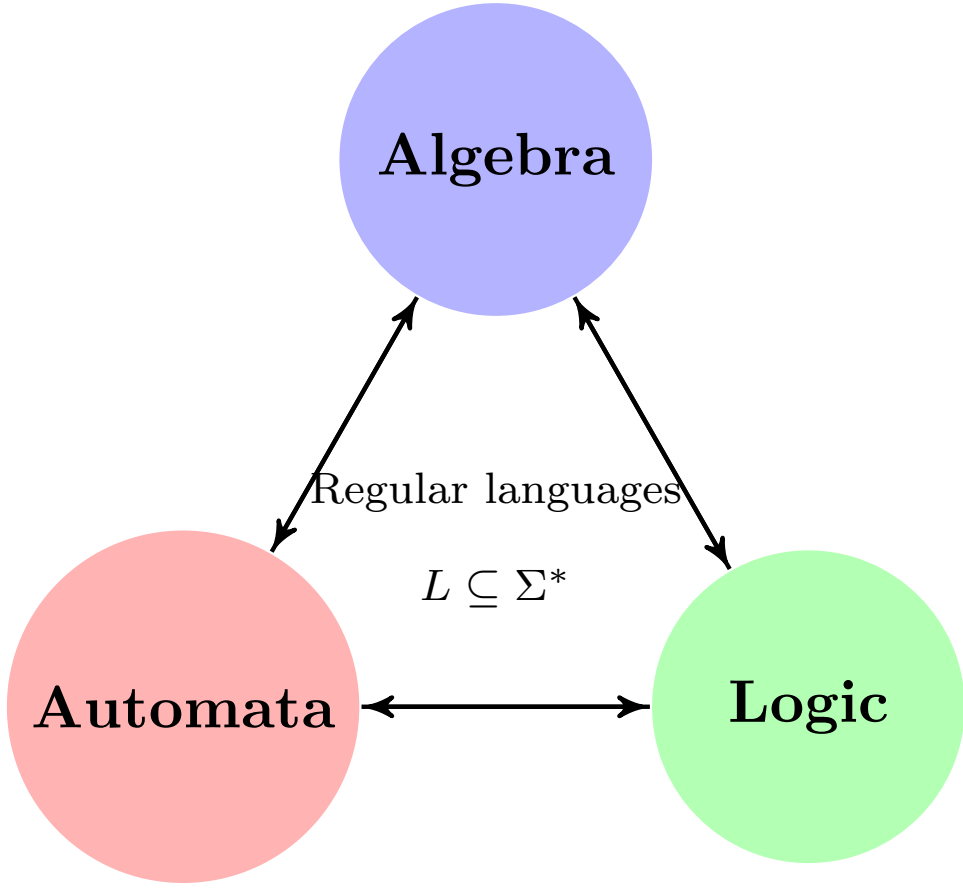
Automata and Logic for (Finite) Word Transductions

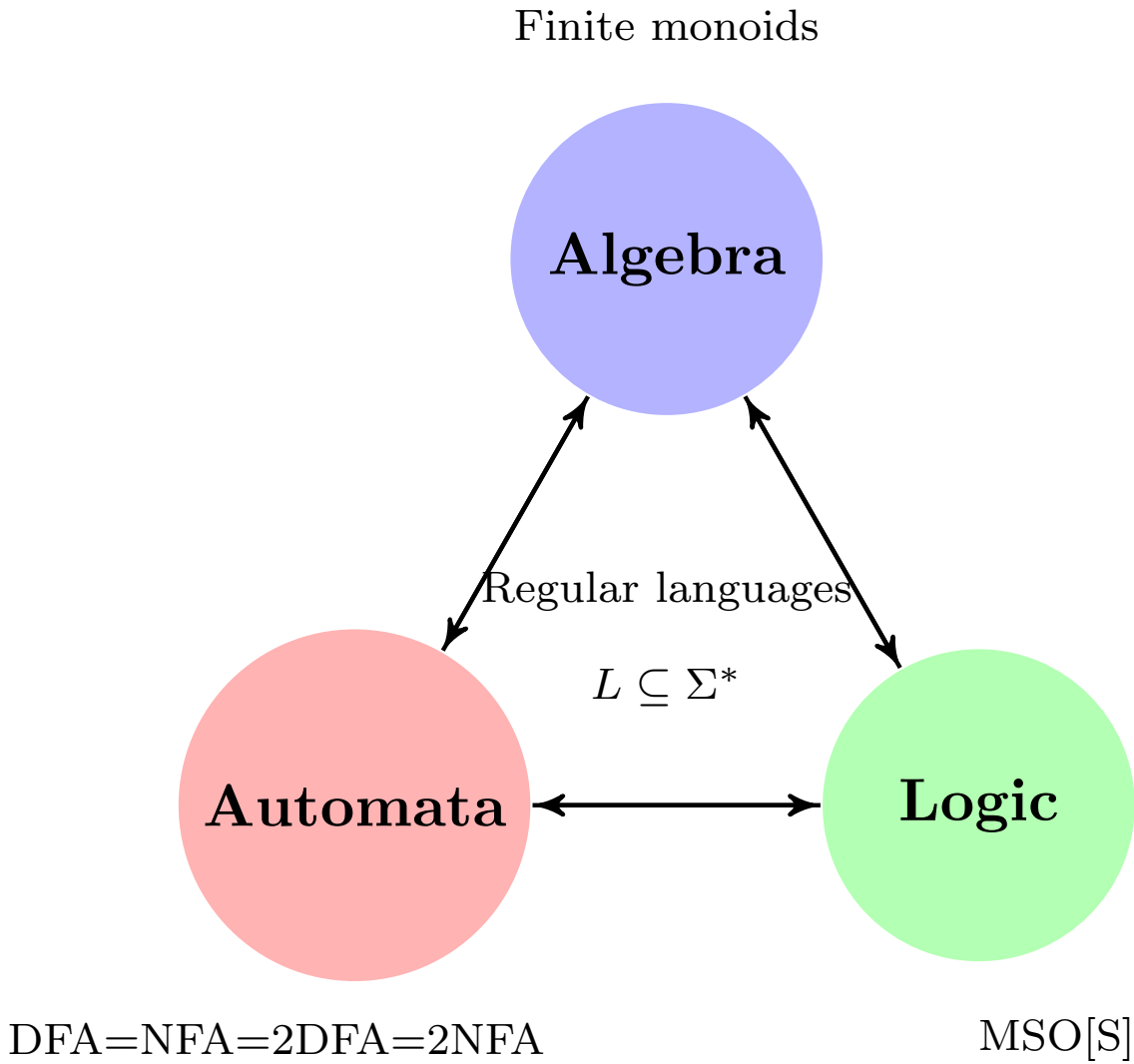
Emmanuel Filiot

Université libre de Bruxelles & FNRS

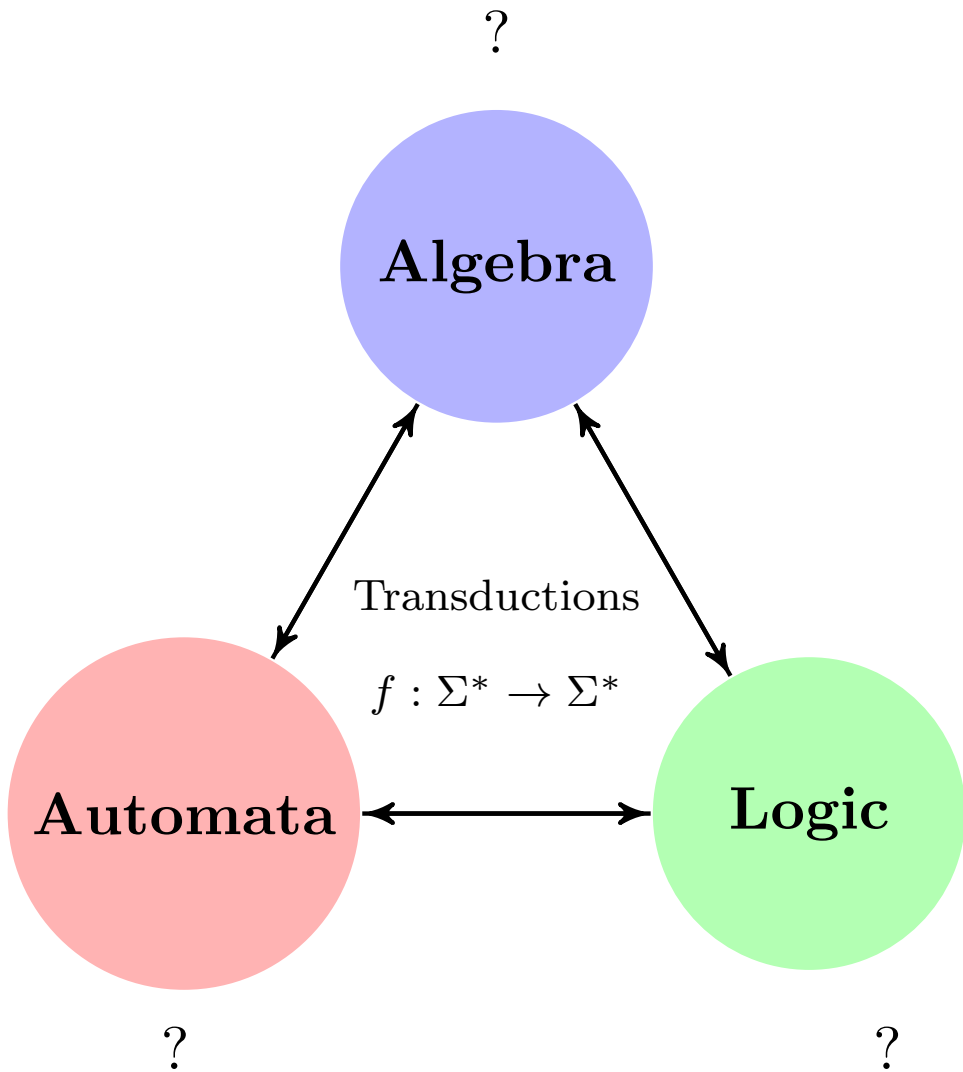
Trinity for Regular Languages



Trinity for Regular Languages



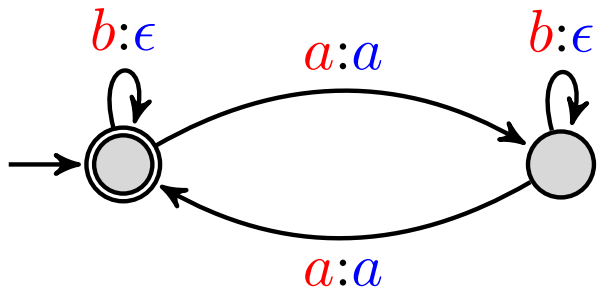
Objective of the talk



Automata models for transductions

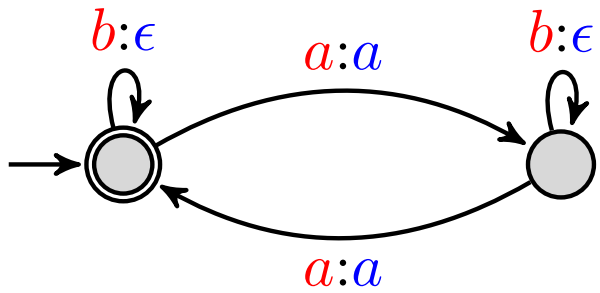
Automata for transductions: transducers

f_{del} :



Automata for transductions: transducers

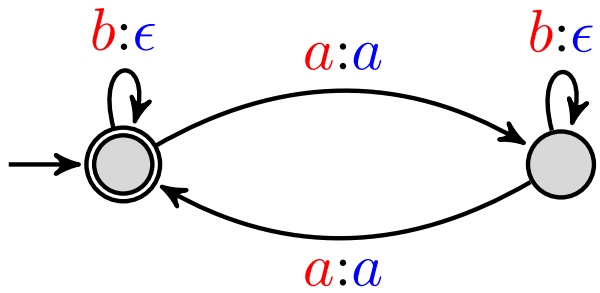
f_{del} :



$$aabaa \quad \mapsto \quad aaaa$$

Automata for transductions: transducers

f_{del} :

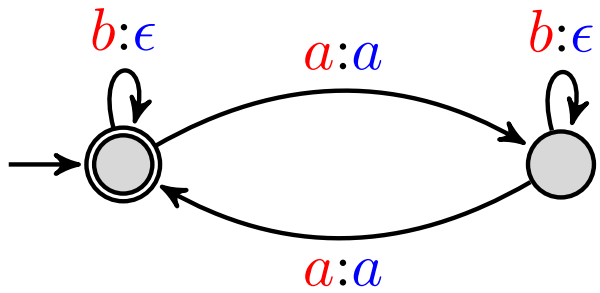


$$aabaa \mapsto aaaa$$

$$aaba \mapsto \text{undefined}$$

Automata for transductions: transducers

f_{del} :

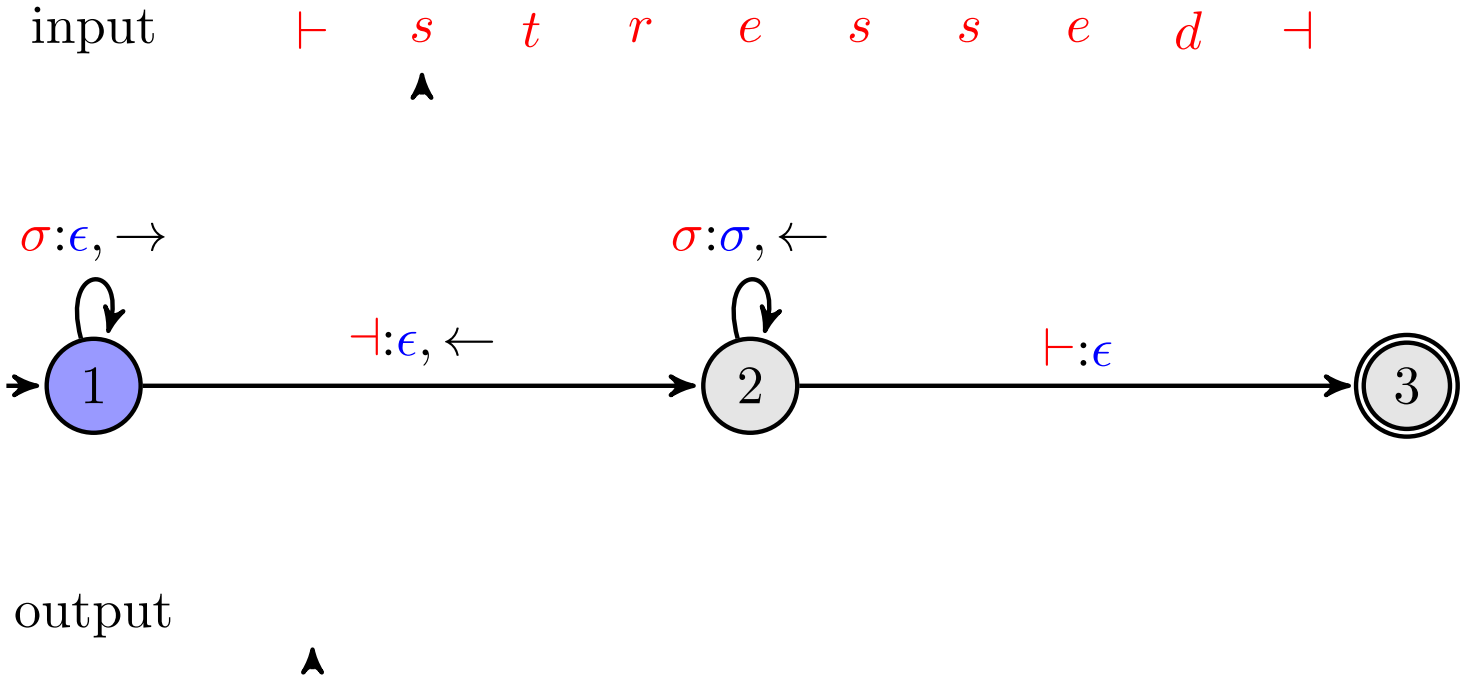


$$aabaa \mapsto aaaa$$

$$aaba \mapsto \text{undefined}$$

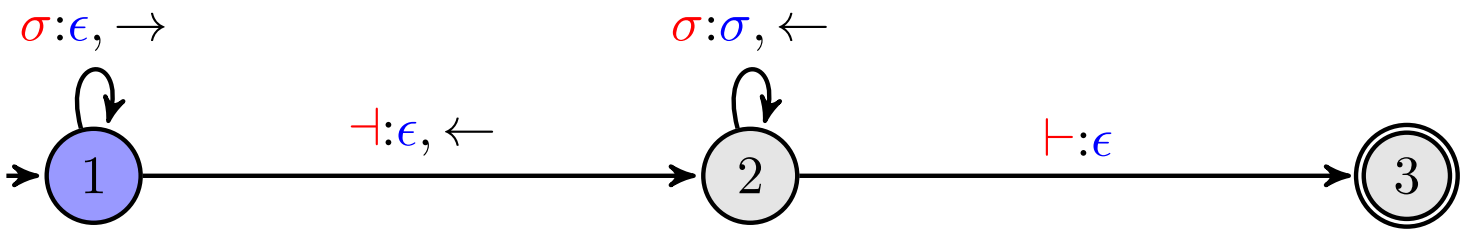
$$dom(f_{del}) = \text{'even number of } a\text{'}$$

Two-way transducers



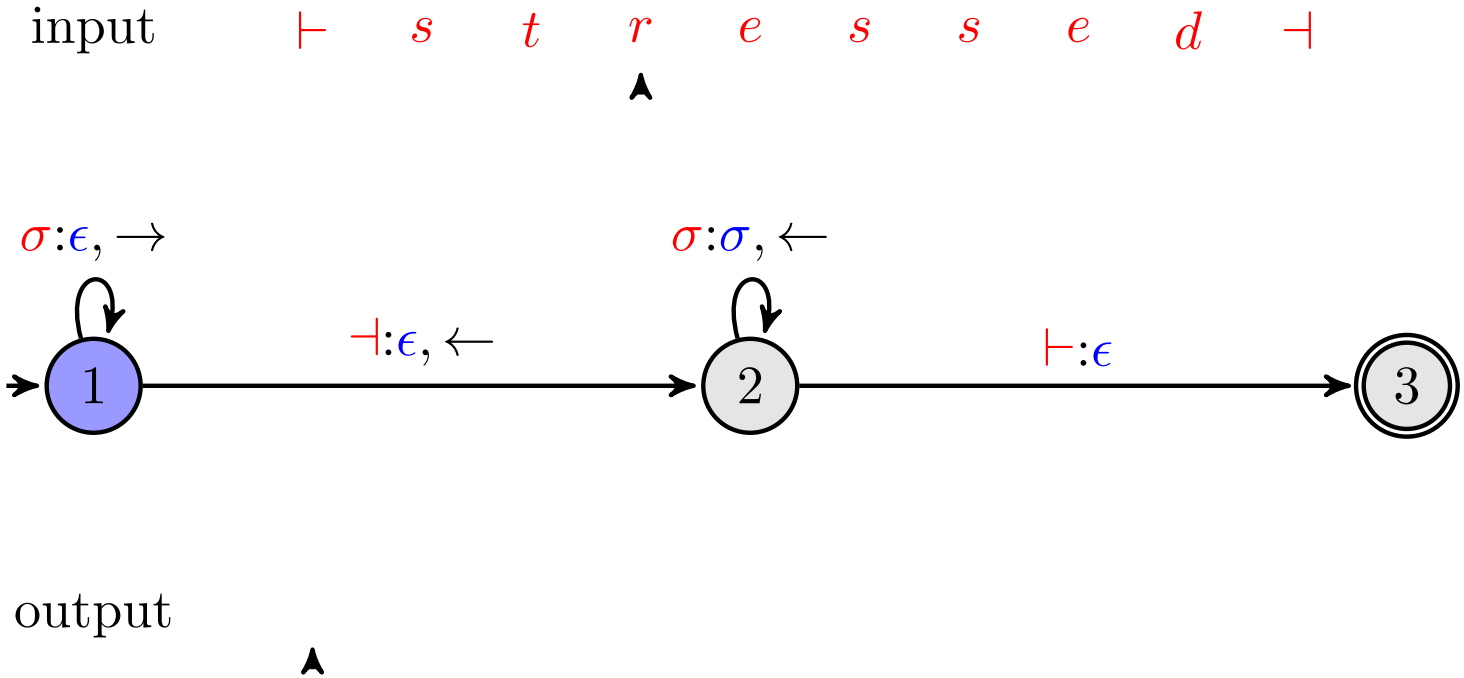
Two-way transducers

input $\vdash$ s t r e s s e d $\dashv$

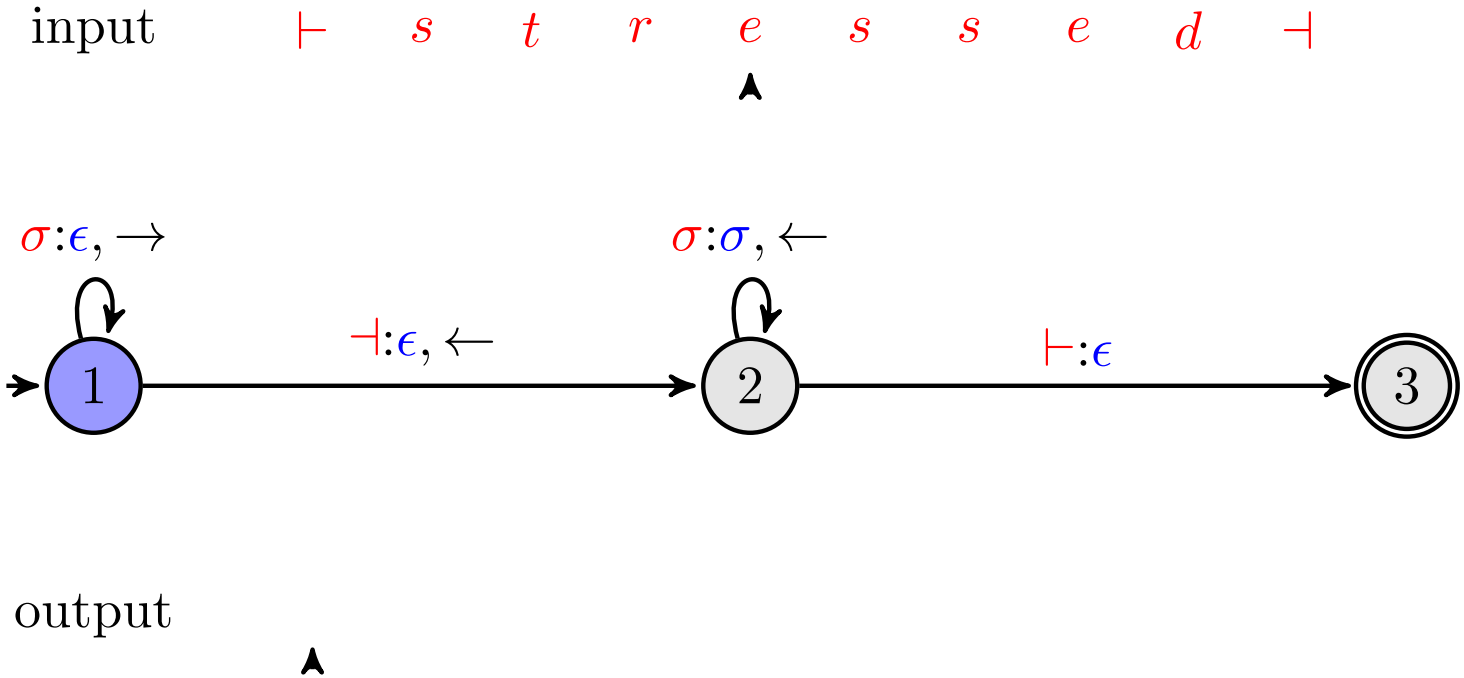


output

Two-way transducers

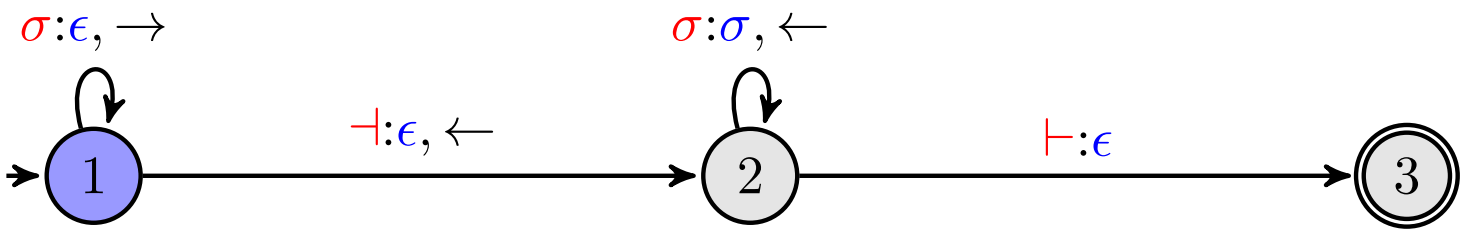


Two-way transducers



Two-way transducers

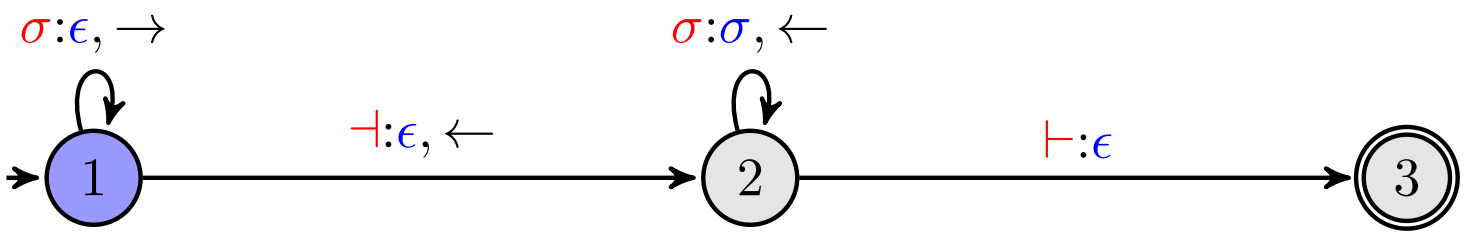
input $\vdash$ s t r e s s e d $\dashv$



output

Two-way transducers

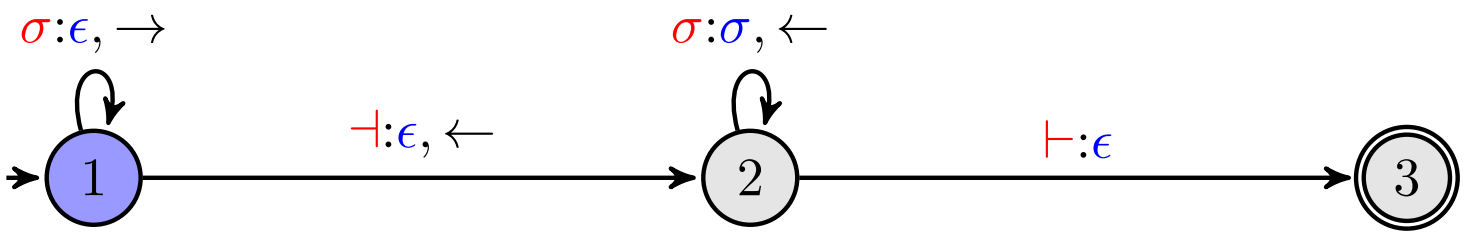
input $\vdash$ s t r e s s e d $\dashv$



output

Two-way transducers

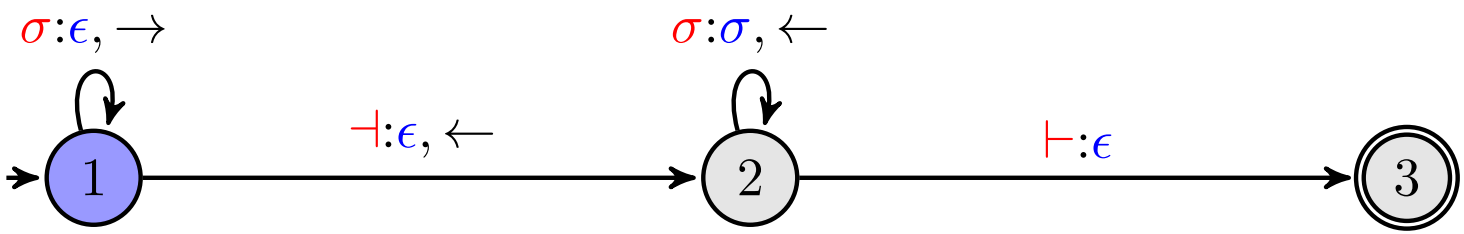
input $\vdash$ s t r e s s e d $\dashv$



output

Two-way transducers

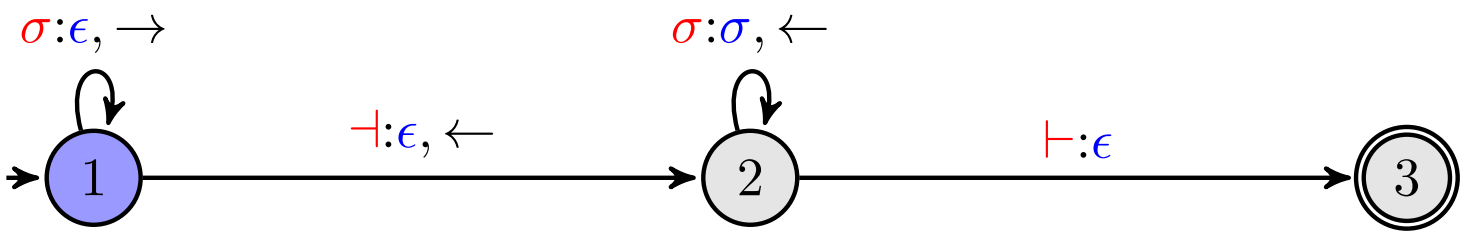
input $\vdash$ s t r e s s e d $\dashv$



output

Two-way transducers

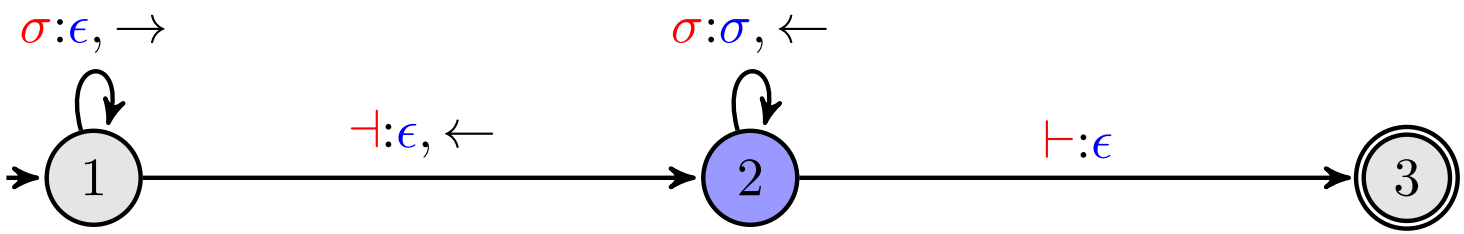
input $\vdash$ s t r e s s e d $\dashv$



output

Two-way transducers

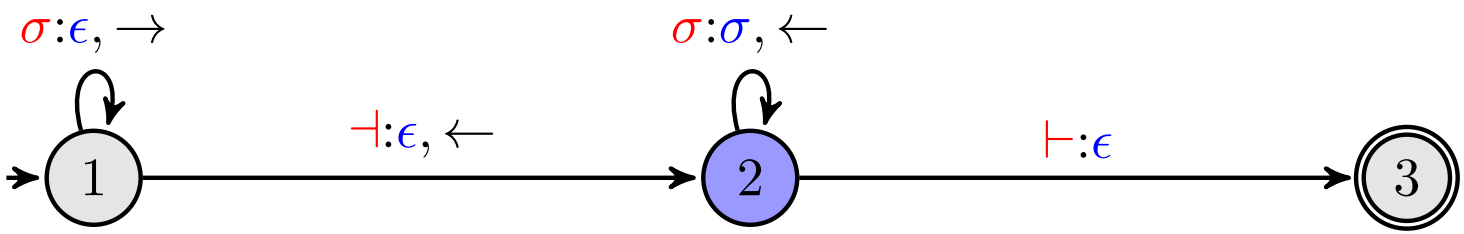
input $\vdash$ s t r e s s e d $\dashv$



output

Two-way transducers

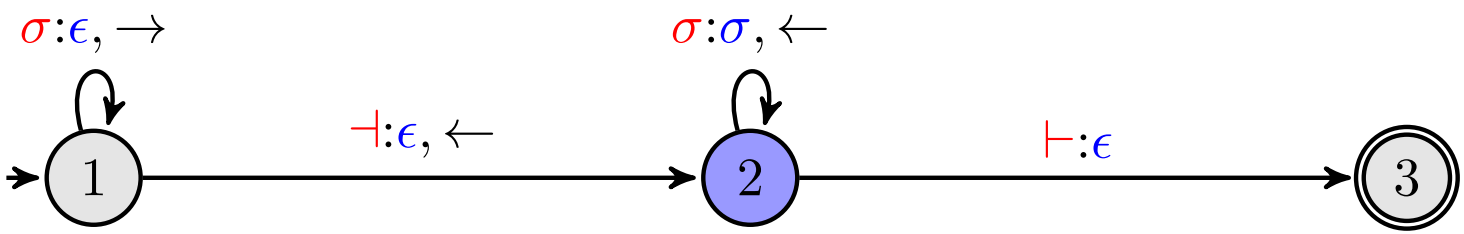
input $\vdash$ s t r e s s e d $\dashv$



output d

Two-way transducers

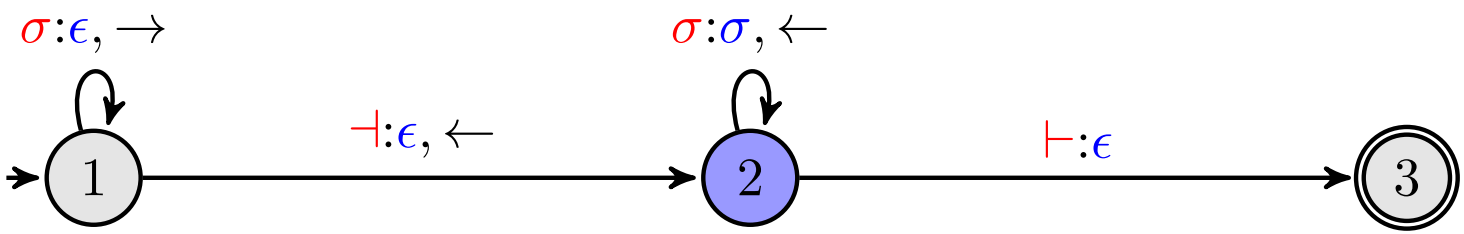
input $\vdash$ s t r e s s e d $\dashv$



output d e

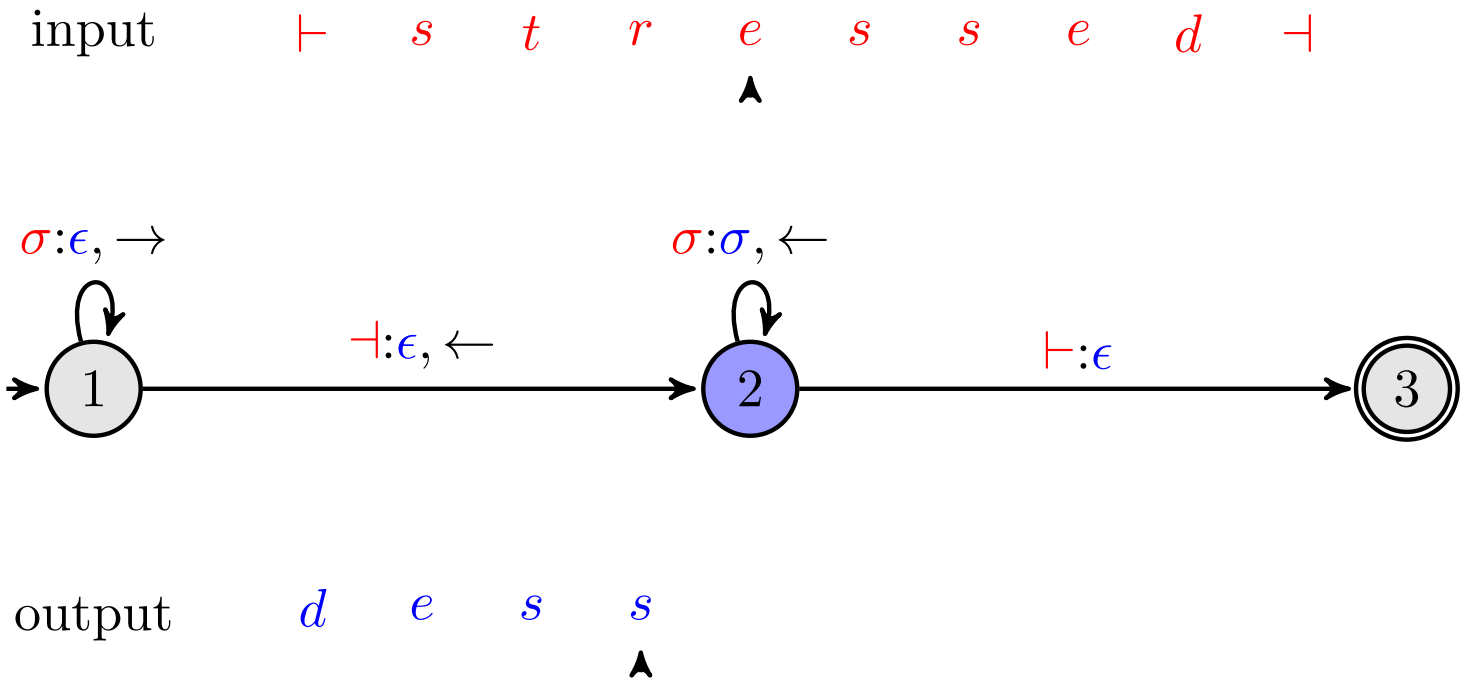
Two-way transducers

input $\vdash$ s t r e s s e d $\dashv$

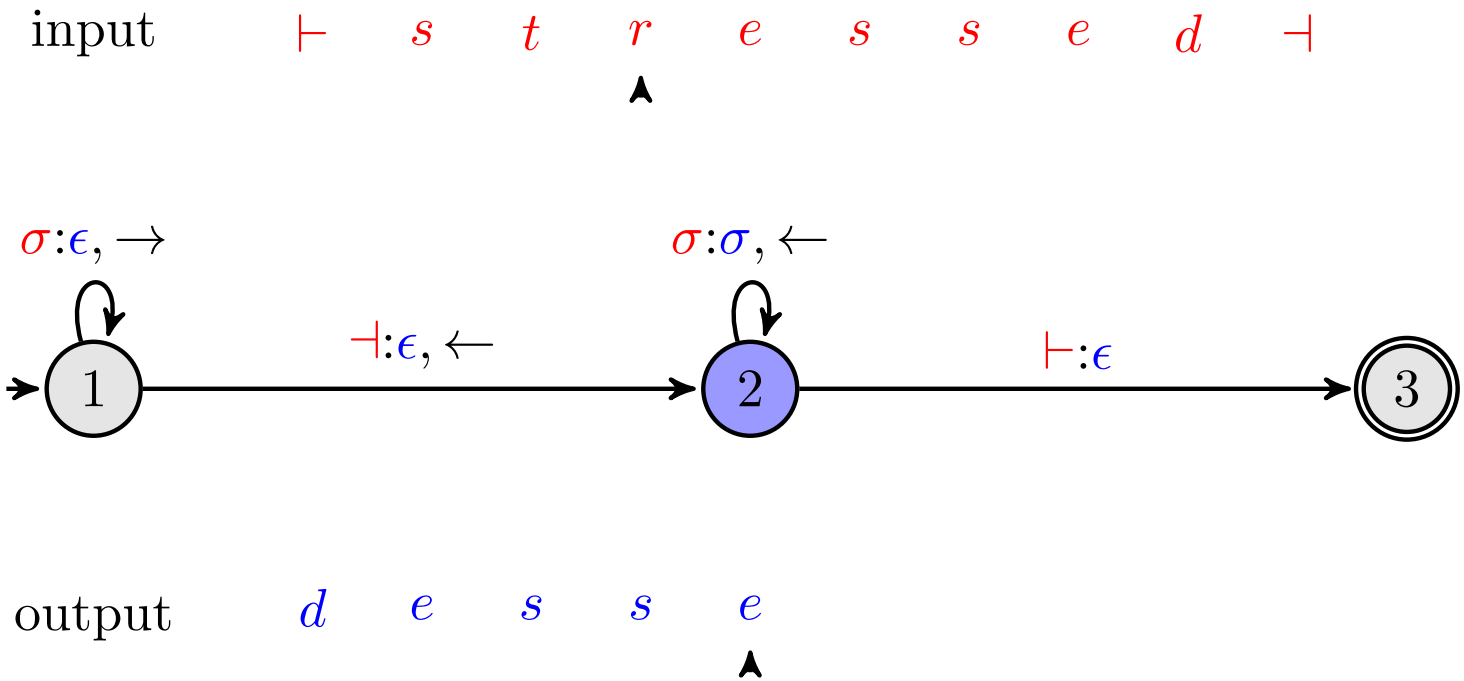


output d e s

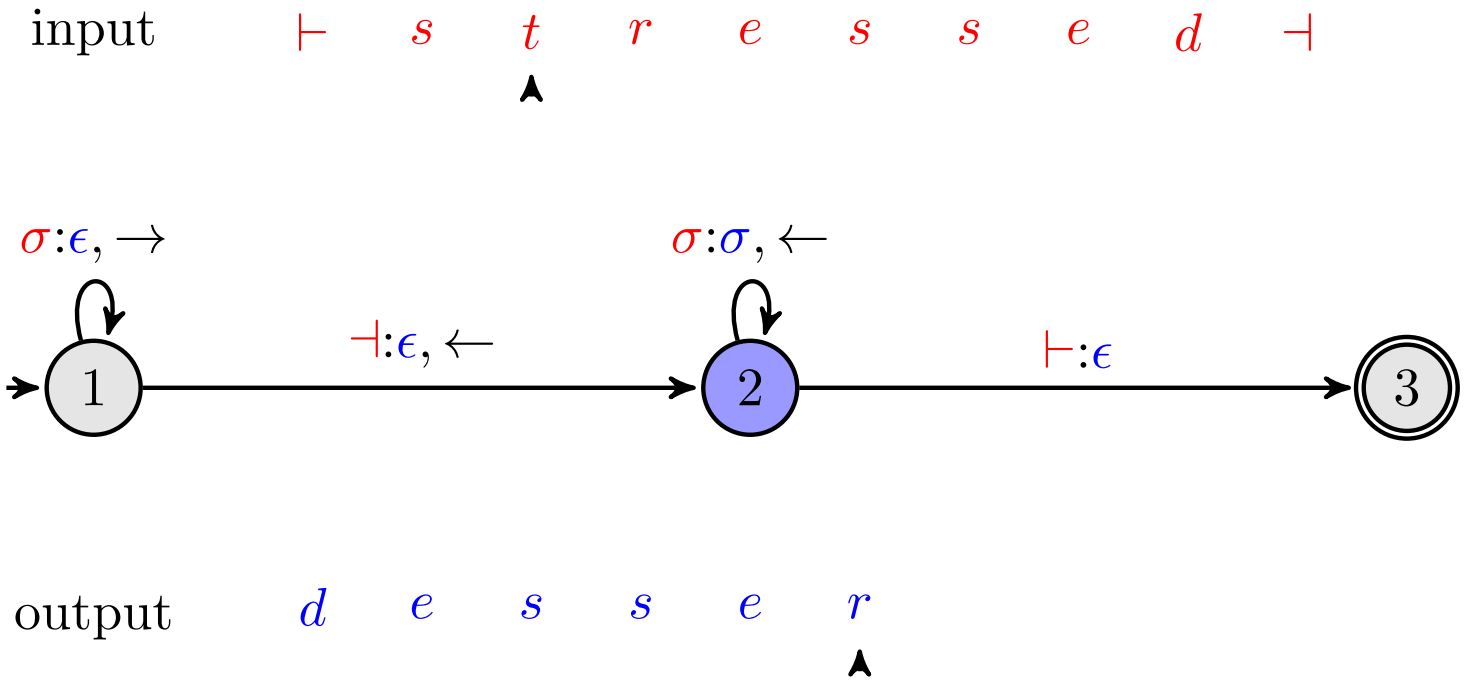
Two-way transducers



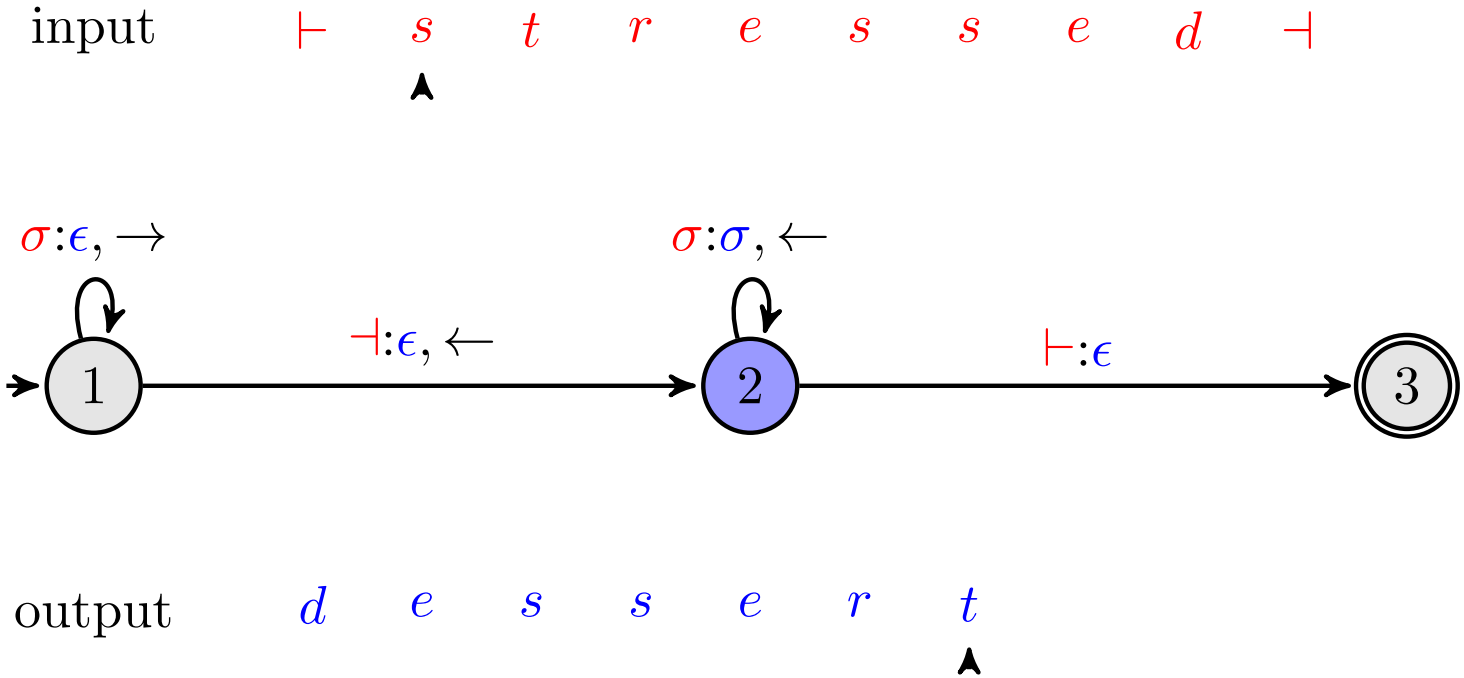
Two-way transducers



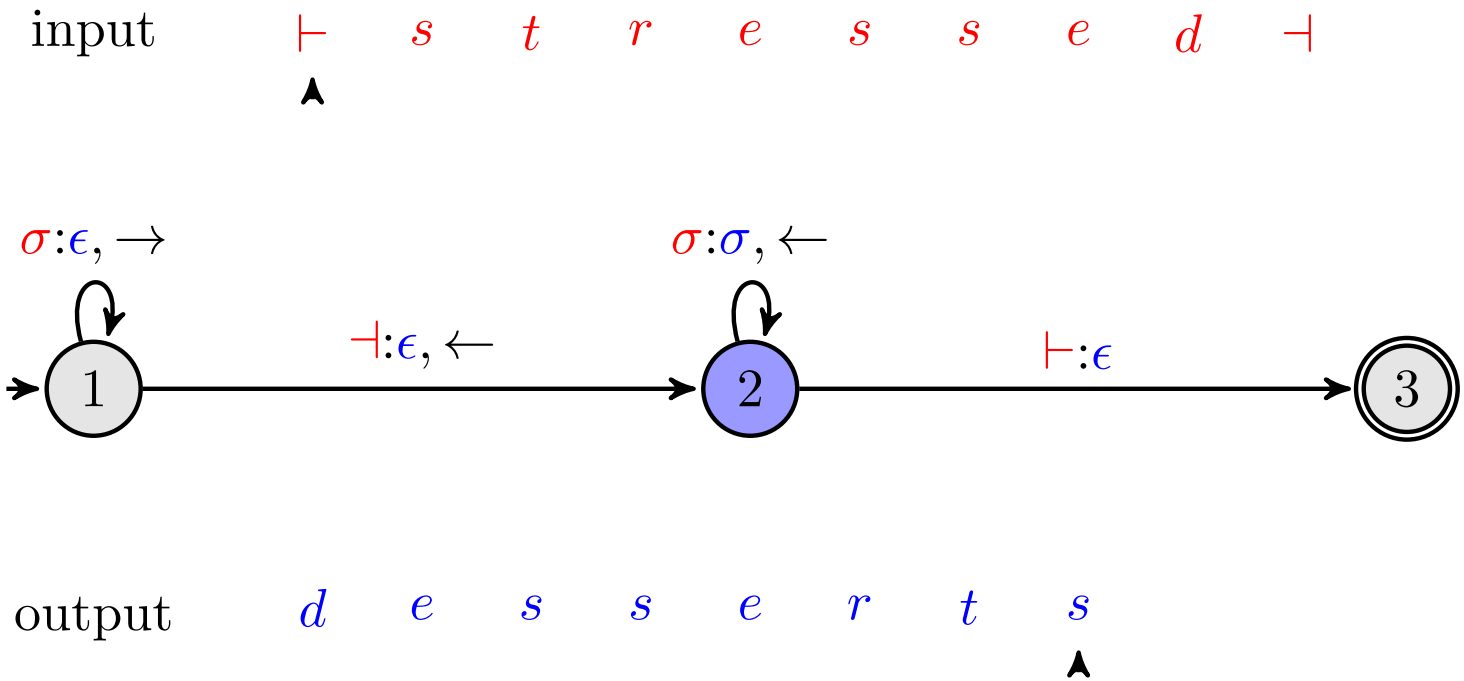
Two-way transducers



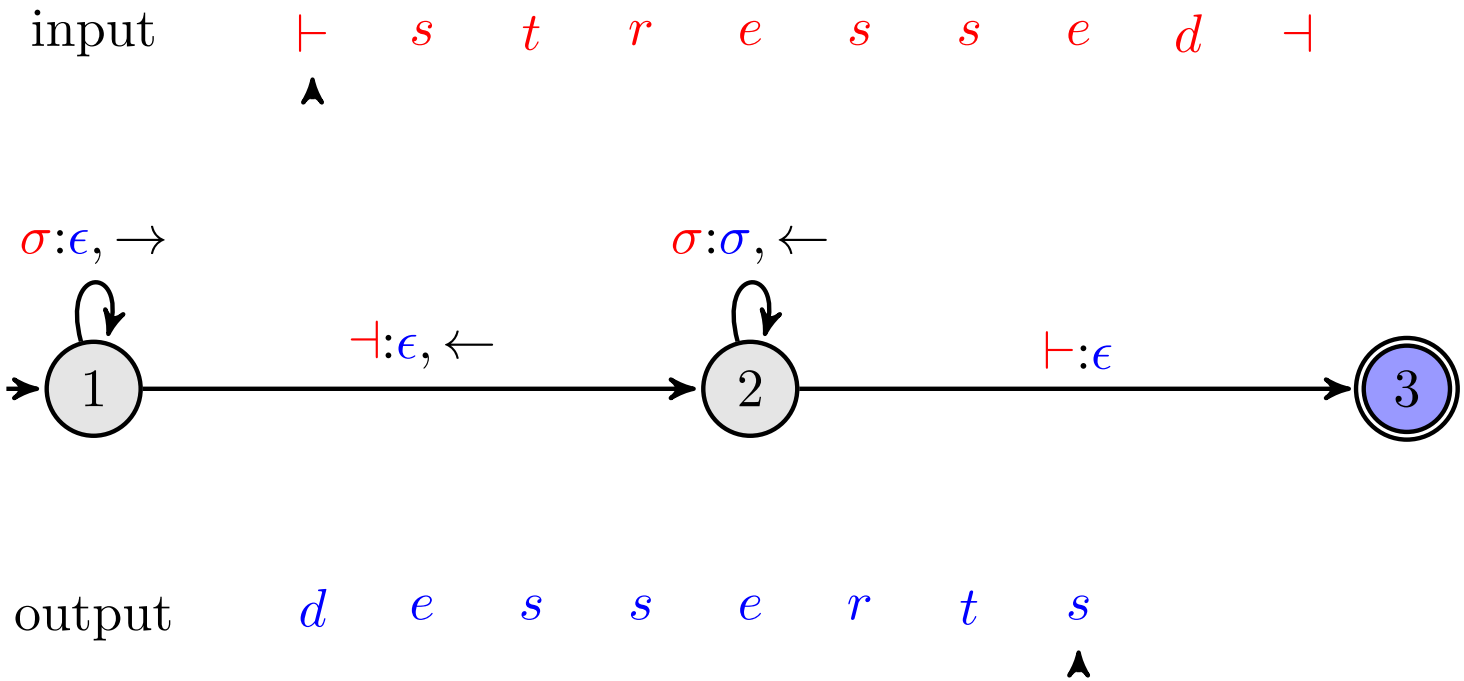
Two-way transducers



Two-way transducers

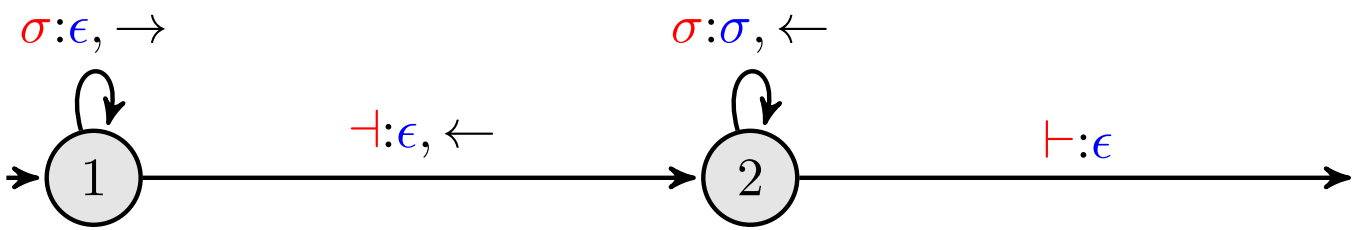


Two-way transducers



Two-way transducers

input $s\ t\ r\ e\ s\ s\ e\ d\ \dashv$



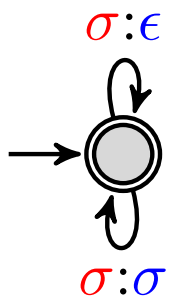
output $d\ e\ s\ s\ e\ r\ t\ s$

one-way $<$ two-way

- ☺ decidable equivalence problem (Culik, Karhumaki, 87).
- ☺ closed under composition $\circ$ (Chytil, Jakl, 77) (Dartois, Fournier, Jecker, Lhote, 17)

Non-determinism and relations

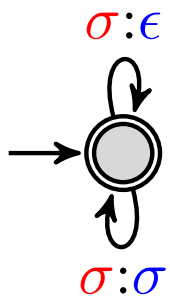
In general, transducers define binary *relations* in $\Sigma^* \times \Sigma^*$



realizes $\{(u, v) \mid v \text{ is a subword of } u\}$

Non-determinism and relations

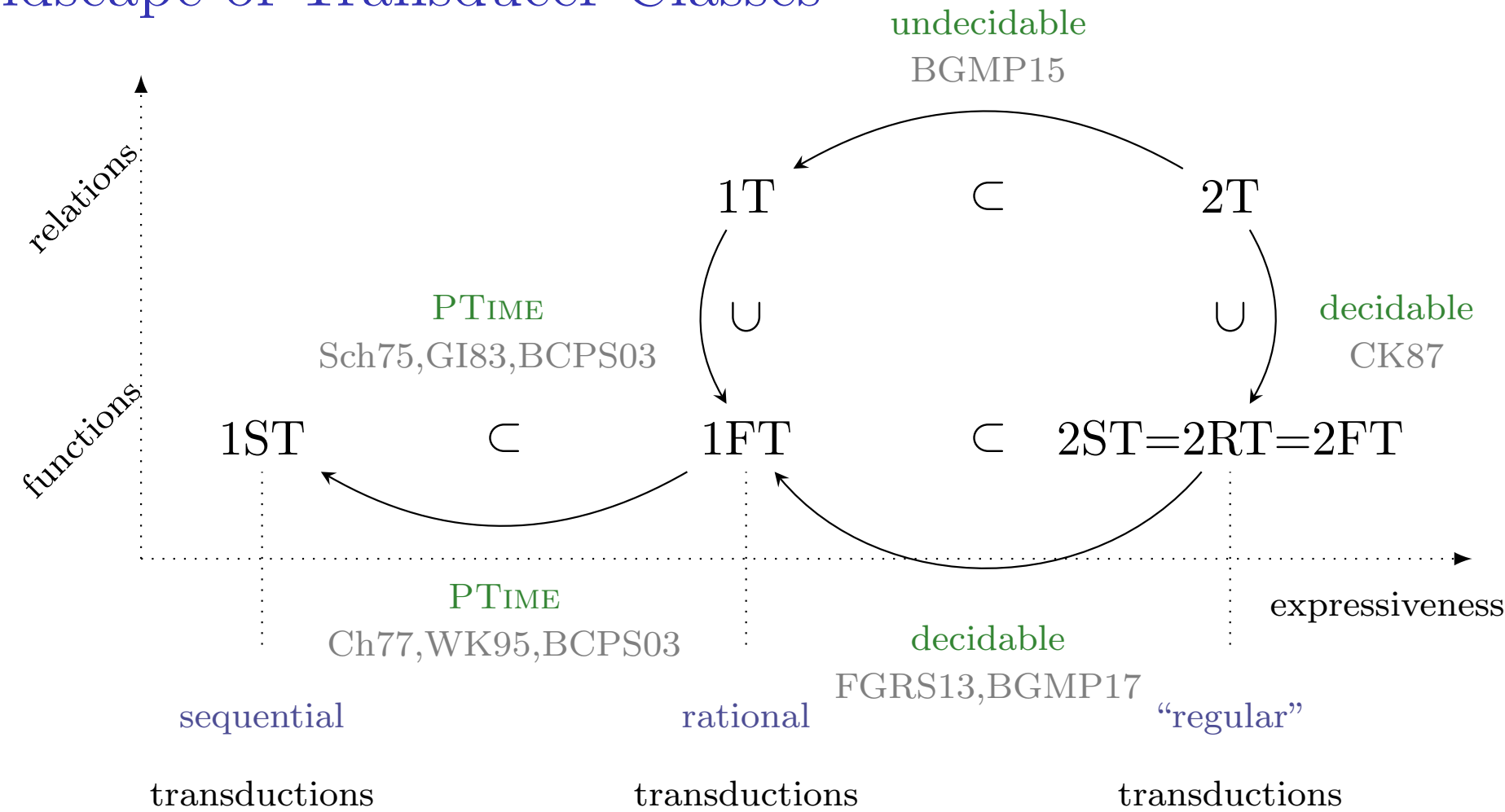
In general, transducers define binary *relations* in $\Sigma^* \times \Sigma^*$



realizes $\{(u, v) \mid v \text{ is a subword of } u\}$

- undecidable equivalence/inclusion problems for non-det 1-way transducers
- sequential transducer $=_{def}$ input-deterministic transducer

Landscape of Transducer Classes[4]



[4] **Sch**:Schutzenberger; **GI**:Gurari, Ibarra; **BCPS**:Beal, Carton, Prieur, Sakarovitch; **Ch**:Choffrut, **WK**:Weber, Klemm; **FGRS**:., Gauwin, Reynier, Servais; **BGMP**:Baschenis,Gauwin,Muscholl,Puppis, **CK**:Culik,Karhumaki

Logic for transductions

(Courcelle) MSO Transducers

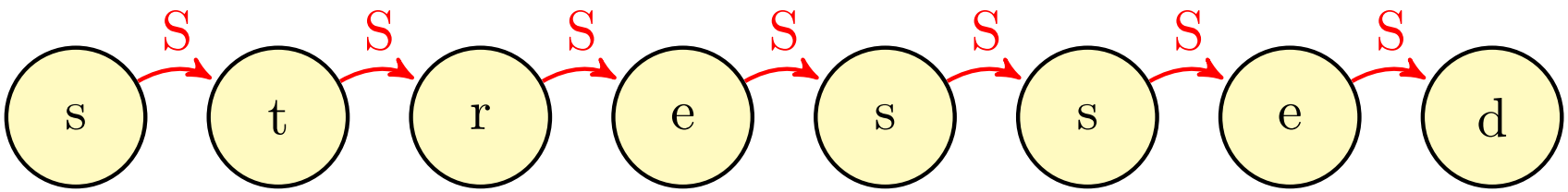
"interpreting the output structure in the input structure"

- output predicates defined by MSO[S] formulas interpreted over the input structure

(Courcelle) MSO Transducers

"interpreting the output structure in the input structure"

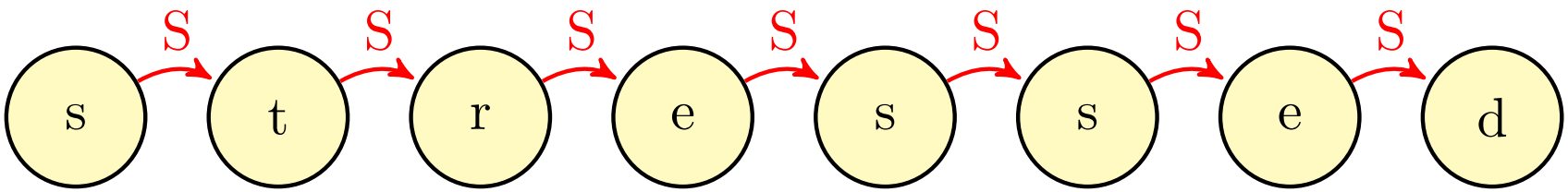
- output predicates defined by MSO[S] formulas interpreted over the input structure



(Courcelle) MSO Transducers

"interpreting the output structure in the input structure"

- output predicates defined by MSO[S] formulas interpreted over the input structure



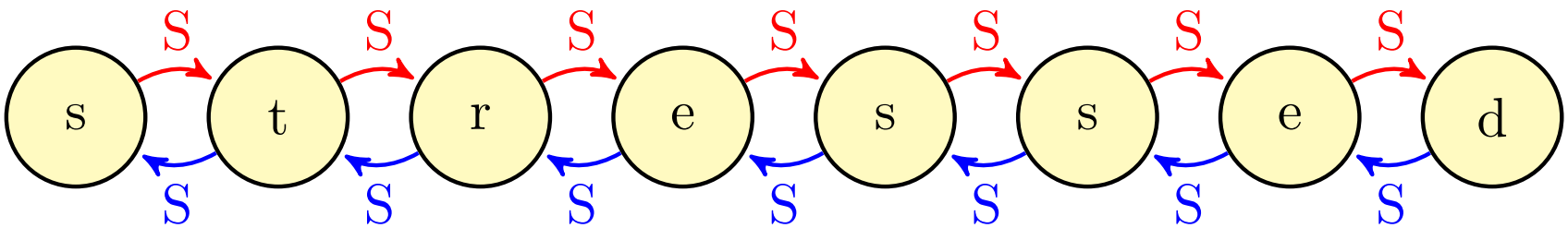
$$\phi_S(x,y) \equiv S(y,x)$$

$$\phi_\sigma(x) \equiv \sigma(x)$$

(Courcelle) MSO Transducers

"interpreting the output structure in the input structure"

- output predicates defined by MSO[S] formulas interpreted over the input structure



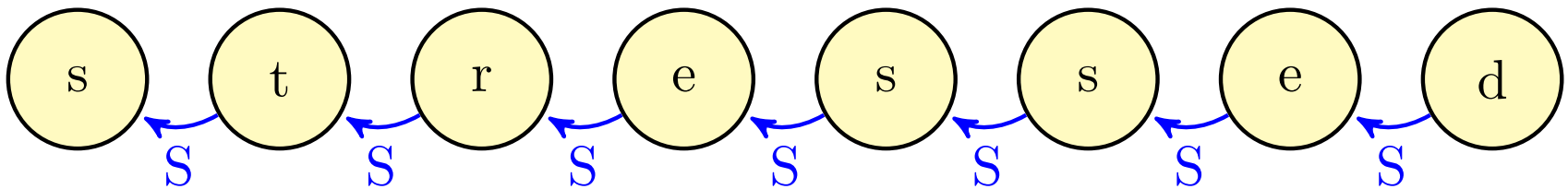
$$\phi_S(x, y) \equiv S(y, x)$$

$$\phi_\sigma(x) \equiv \sigma(x)$$

(Courcelle) MSO Transducers

"interpreting the output structure in the input structure"

- output predicates defined by MSO[S] formulas interpreted over the input structure



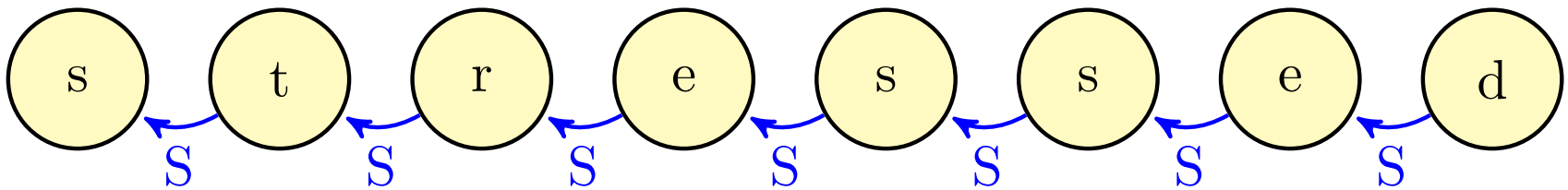
$$\phi_S(x,y) \equiv S(y,x)$$

$$\phi_\sigma(x) \equiv \sigma(x)$$

(Courcelle) MSO Transducers

"interpreting the output structure in the input structure"

- output predicates defined by MSO[S] formulas interpreted over the input structure

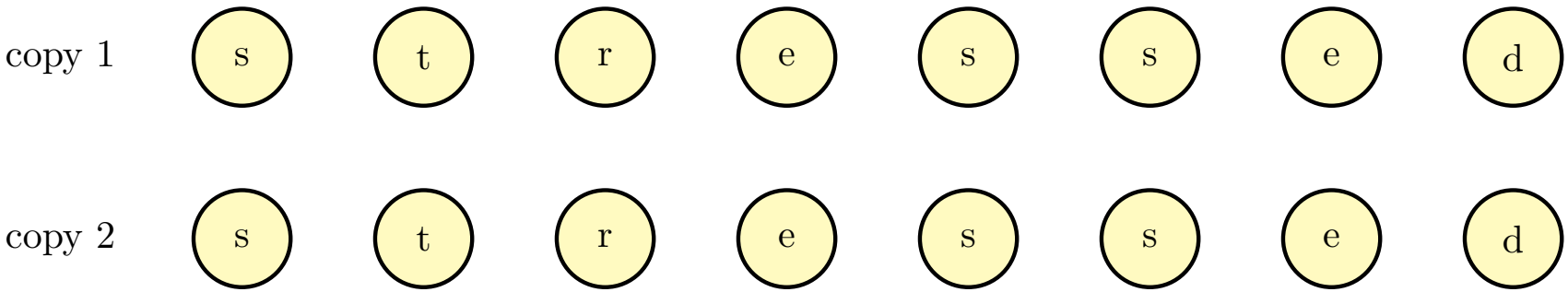


$$\phi_S(x,y) \equiv S(y,x)$$

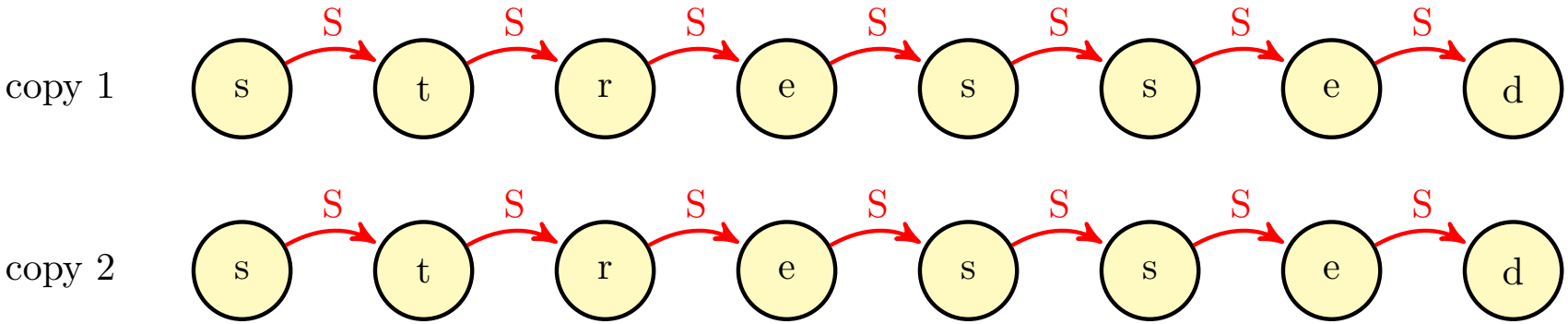
$$\phi_\sigma(x) \equiv \sigma(x)$$

- input structure can be copied a fixed number of times: $u \mapsto uu$, or $u \mapsto u.\mathrm{mirror}(u)$.

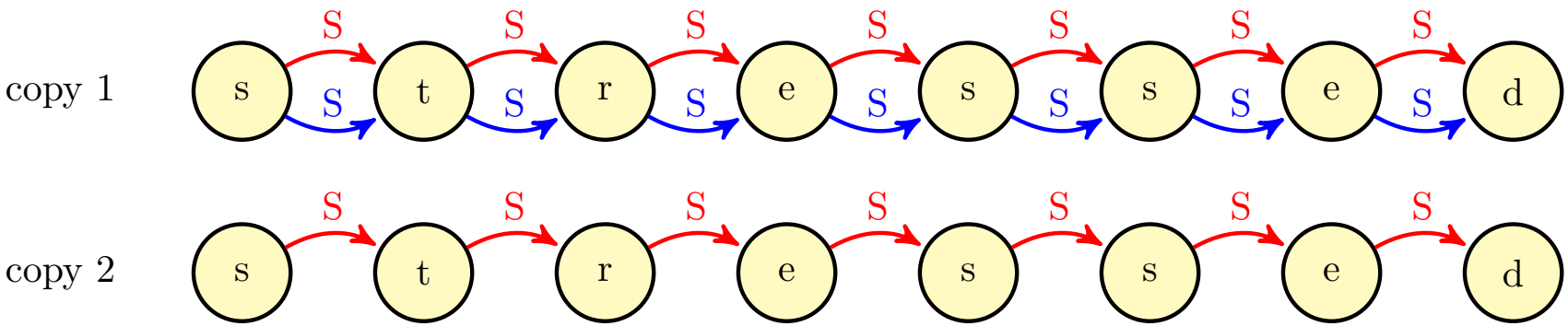
Other example : $u \mapsto u.\mathrm{mirror}(u)$



Other example : $u \mapsto u.\mathrm{mirror}(u)$



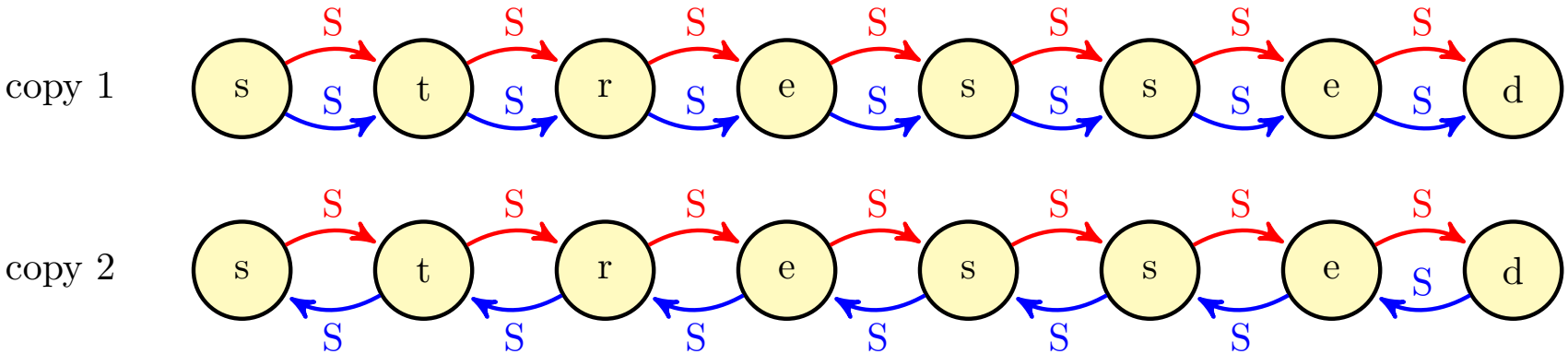
Other example : $u \mapsto u.\mathrm{mirror}(u)$



Formulas

copy 1: $\phi_S^{\mathbf{1}}(x,y) \equiv S(x,y)$

Other example : $u \mapsto u.\mathrm{mirror}(u)$

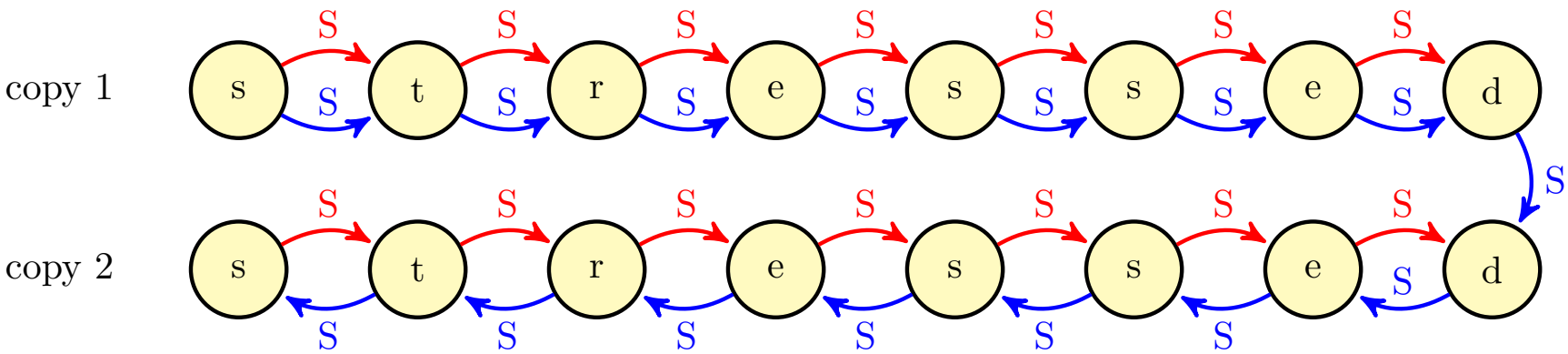


Formulas

$$\textbf{copy 1:} \qquad \phi_S^{\mathbf{1}}(x,y) \equiv S(x,y)$$

$$\textbf{copy 2:} \qquad \phi_S^{\mathbf{2}}(x,y) \equiv S(y,x)$$

Other example : $u \mapsto u.\mathrm{mirror}(u)$



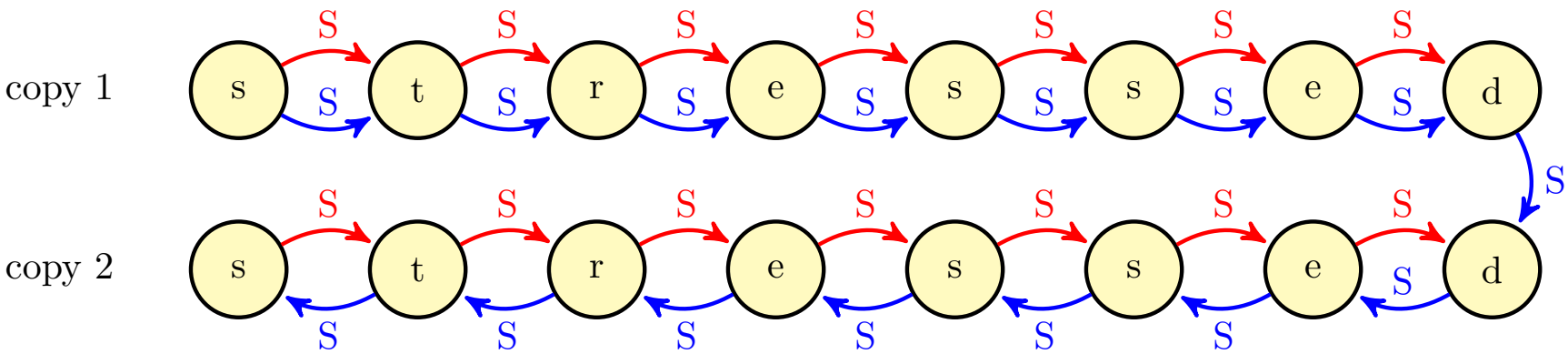
Formulas

$$\textbf{copy 1:} \quad \phi_S^{\mathbf{1}}(x,y) \equiv S(x,y)$$

$$\textbf{copy 2:} \quad \phi_S^{\mathbf{2}}(x,y) \equiv S(y,x)$$

$$\textbf{copy 1 to copy 2:} \quad \phi_S^{\mathbf{1}\to\mathbf{2}}(x,y) \equiv x = y \wedge last(x)$$

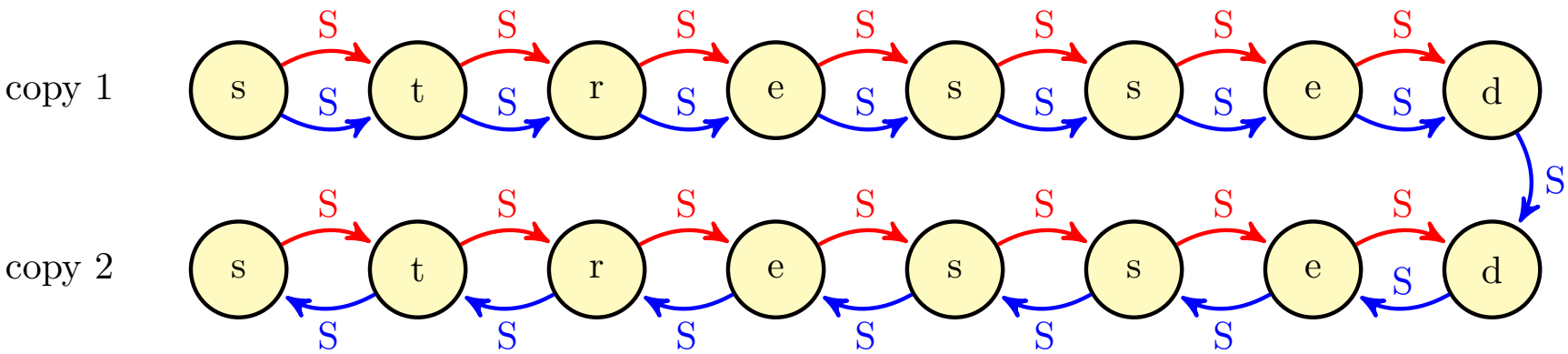
Other example : $u \mapsto u.\mathrm{mirror}(u)$



Formulas

$$\begin{aligned}
\textbf{copy 1:} \quad & \phi_S^{\mathbf{1}}(x,y) \equiv S(x,y) \\
\textbf{copy 2:} \quad & \phi_S^{\mathbf{2}}(x,y) \equiv S(y,x) \\
\textbf{copy 1 to copy 2:} \quad & \phi_S^{\mathbf{1}\to\mathbf{2}}(x,y) \equiv x = y \wedge last(x) \\
\textbf{copy 2 to copy 1:} \quad & \phi_S^{\mathbf{2}\to\mathbf{1}}(x,y) \equiv \bot
\end{aligned}$$

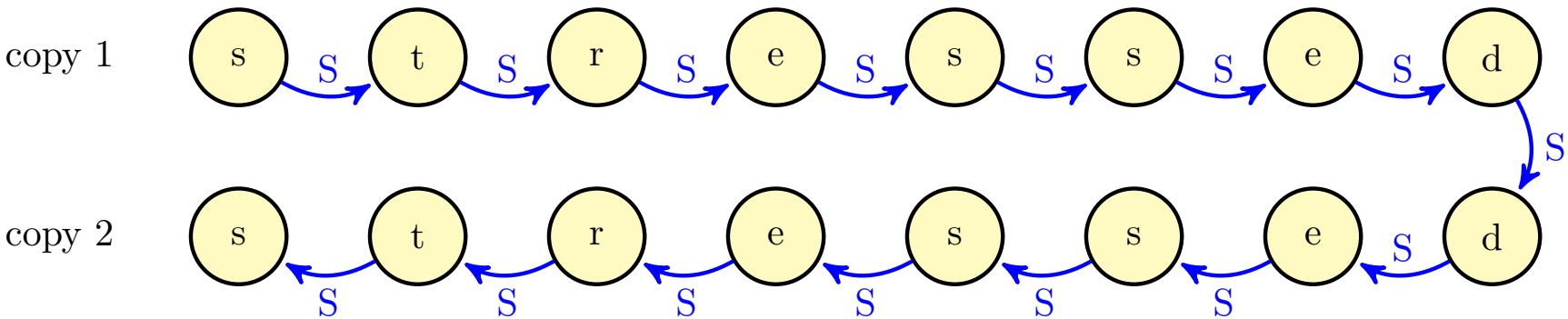
Other example : $u \mapsto u.\mathrm{mirror}(u)$



Formulas

$$\begin{aligned}
\textbf{copy 1:} \quad & \phi_S^{\mathbf{1}}(x,y) \equiv S(x,y) \\
\textbf{copy 2:} \quad & \phi_S^{\mathbf{2}}(x,y) \equiv S(y,x) \\
\textbf{copy 1 to copy 2:} \quad & \phi_S^{\mathbf{1}\to\mathbf{2}}(x,y) \equiv x = y \wedge last(x) \\
\textbf{copy 2 to copy 1:} \quad & \phi_S^{\mathbf{2}\to\mathbf{1}}(x,y) \equiv \bot \\
\textbf{for all copies } i\textbf{:} \quad & \phi_\sigma^{\mathbf{i}}(x) \equiv \sigma(x)
\end{aligned}$$

Other example : $u \mapsto u.\mathrm{mirror}(u)$



Formulas

copy 1:	$\phi_S^{\mathbf{1}}(x,y)$	$\equiv$	$S(x,y)$
copy 2:	$\phi_S^{\mathbf{2}}(x,y)$	$\equiv$	$S(y,x)$
copy 1 to copy 2:	$\phi_S^{\mathbf{1}\to\mathbf{2}}(x,y)$	$\equiv$	$x = y \wedge last(x)$
copy 2 to copy 1:	$\phi_S^{\mathbf{2}\to\mathbf{1}}(x,y)$	$\equiv$	$\bot$
for all copies i:	$\phi_\sigma^{\mathbf{i}}(x)$	$\equiv$	$\sigma(x)$

Büchi Theorems for Word Transductions

Let $f : \Sigma^* \to \Sigma^*$.

Theorem (Engelfriet, Hoogeboom, 01)

f is 2ST-definable iff f is MSO-definable.

Büchi Theorems for Word Transductions

Let $f : \Sigma^* \to \Sigma^*$.

Theorem (Engelfriet, Hoogeboom, 01)

f is 2ST-definable iff f is MSO-definable.

Consequence Equivalence is decidable for MSO-transducers, and they are closed under composition.

A DECIDABLE LOGIC FOR TRANSDUCTIONS OF FINITE WORDS

Joint with Luc Dartois ULB
Nathan Lhote ULB - LABRI

MOTIVATION

MODEL-CHECKING :

IMPLEMENTATION $\models$ SPECIFICATION ?

↑ A functional transduction $f : \Sigma^* \hookrightarrow \Sigma^*$

↑ A transduction $R \subseteq \Sigma^* \times \Sigma^*$

$f \models R$ if $\forall u \in dom(f)$, $(u, f(u)) \in R$

MOTIVATION

MODEL-CHECKING :

IMPLEMENTATION $\models$ SPECIFICATION ?

↑ A functional transduction $f : \Sigma^* \hookrightarrow \Sigma^*$

↑ A transduction $R \subseteq \Sigma^* \times \Sigma^*$

$f \models R$ if $\forall u \in dom(f),\ (u, f(u)) \in R$

SYNTHESIS :

? $\models$ SPECIFICATION

$\exists f \cdot\ f \models R \ \wedge\ dom(f) = dom(R)$?

EXAMPLES OF SPECIFICATIONS

- True: Σ^*, Σ^*

EXAMPLES OF SPECIFICATIONS

- True: $\Sigma^* \times \Sigma^*$
- There exists an 'a' in the output $\Sigma^* \times \Sigma_a^* \Sigma^*$

EXAMPLES OF SPECIFICATIONS

- True: $\Sigma^* \times \Sigma^*$
- There exists an 'a' in the output $\Sigma^* \times \Sigma^* a \Sigma^*$
- Output is a subword of input

$$\{ (u, v) \mid \exists i_1 < \dots < i_{|v|} \cdot \quad v = u[i_1] \dots u[i_{|v|}] \}$$

EXAMPLES OF SPECIFICATIONS

- True: $\Sigma^* \times \Sigma^*$
- There exists an 'a' in the output $\Sigma^* \times \Sigma^* a \Sigma^*$
- Output is a subword of input

  $\{ (u, v) \mid \exists i_1 < \dots < i_{|v|} \cdot \; v = u[i_1] \dots u[i_{|v|}] \}$
- Every request is processed exactly once

  $\{ (r_1 \cdots r_k, g_{\pi(1)} \cdots g_{\pi(k)}) \mid \pi \text{ permutation} \}$

EXAMPLES OF SPECIFICATIONS

- True: $\Sigma^* \times \Sigma^*$
- There exists an 'a' in the output $\Sigma^* \times \Sigma^* a \Sigma^*$
- Output is a subword of input

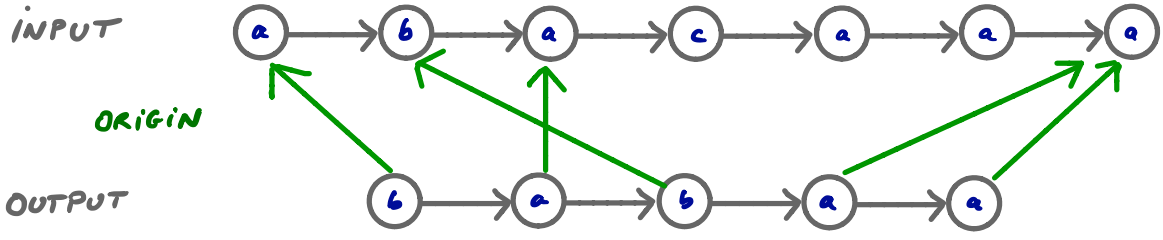
  $\{ (u, v) \mid \exists i_1 < \dots < i_{|v|} \cdot \; v = u[i_1] \dots u[i_{|v|}] \}$
- Every request is processed exactly once

  $\{ (r_1 \cdots r_k, g_{\pi(1)} \cdots g_{\pi(k)}) \mid \pi \text{ permutation} \}$
- More generally: shuffle

  $\{ (\sigma_1 \cdots \sigma_k, \sigma_{\pi(1)} \cdots \sigma_{\pi(k)}) \mid \pi \text{ permutation} \}$

TOWARDS A LOGIC FOR TRANSDUCTIONS

IDEA : SEE TRANSDUCTIONS AS SINGLE STRUCTURES WITH ORIGIN
(called origin graphs)

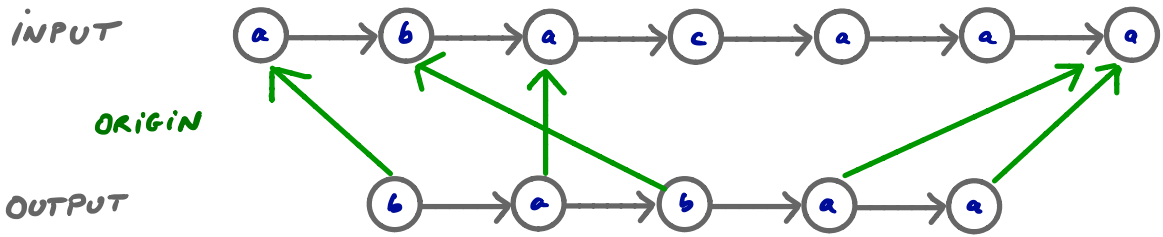


$$MSO[\leq_{in}, \leq_{out}, \theta]$$

input order — output order — origin function

TOWARDS A LOGIC FOR TRANSDUCTIONS

IDEA : SEE TRANSDUCTIONS AS SINGLE STRUCTURES WITH ORIGIN (called origin graphs)



$$MSO[\leq_{in}, \leq_{out}, \theta]$$

$\leq_{in}$: input order

$\leq_{out}$: output order

θ : origin function

Origin information [Bojańczyk, '14]

Characterizations of classes of origin graphs [Bojańczyk, Daviaud, Guillon, Penelle '17]

EXAMPLES

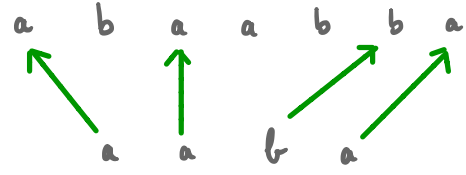
- True : T

EXAMPLES

- True : $\top$
- Output contains 'a' : $\exists^{out}\, x.\ a(x)$ $\quad (\exists^{out}\, x =_{def} \exists x.\ x \leq_{out} x)$

EXAMPLES

- True : $\top$
- Output contains 'a' : $\exists^{out} x.\ a(x)$
- Output is a subword of input :



$$\underbrace{\forall^{out}_x \forall^{out}_y\ x \leq_{out} y \rightarrow \sigma(x) \leq_{in} \sigma(y)}_{\sigma \text{ is order-preserving}} \wedge \underbrace{\forall^{out}_x.\ \bigwedge_{\sigma \in \Sigma} a(x) \rightarrow a(\sigma(x))}_{\sigma \text{ is label-preserving}}$$

$$\wedge\ \underbrace{\forall^{out}_x \forall^{out}_y\ x \neq y \rightarrow \sigma(x) \neq \sigma(y)}_{\sigma \text{ is injective}}$$

EXAMPLES

- True : $\top$
- Output contains 'a' : $\exists^{out} x.\ a(x)$
- Output is a subword of input :
- shuffle : $\text{bijection}(\theta) \wedge \text{label-pres}(\theta)$

EXAMPLES

- True : $\top$
- Output contains 'a' : $\exists^{out} x.\ a(x)$
- Output is a subword of input :
- shuffle : $\text{bijection}(\sigma) \wedge \text{label-pres}(\sigma)$
- identity : $\text{shuffle} \wedge \text{order-preserving}(\sigma)$

MODEL-CHECKING

$T \models \varphi$ is decidable for $T \in 2DFT$ and $\varphi \in MSO[\leq_{in}, \leq_{out}, \theta]$

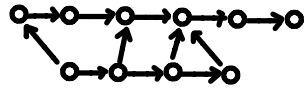
Bojańczyk, Daviaud, Guillon, Penelle '17

MODEL-CHECKING

$T \not\models \varphi$ is decidable for $T \in 2DFT$ and $\varphi \in MSO[\leq_{in}, \leq_{out}, \theta]$

Bojańczyk, Daviaud, Guillon, Penelle '17

WORD $\quad\quad$ SET OF ORIGIN GRAPHS

$u \in \Sigma^* \xrightarrow{\alpha} \left\{ \; \text{  } \; \text{of } \begin{matrix} u \\ v \end{matrix} \;\middle|\; (u,v,\sigma) \in [\![T]\!] \right\}$

1. $\alpha \in$ Courcelle's word-to-graph transductions
2. Backward Translation Theorem: $\alpha^{-1}([\![\neg\varphi]\!]) \in REG(\Sigma)$
3. Check $\alpha^{-1}([\![\neg\varphi]\!]) = \emptyset$

SATISFIABILITY Input : $\varphi \in MSO[\leq_{in}, \leq_{out}, \sigma]$ Output : $\exists G \cdot G \models \varphi$?

RESULTS

- Undecidable (even for $FO_2[\leq_{out}, S_{out}, \leq_{in}, \sigma]$)
- Decidable for $\mathcal{L}_T := FO_2[\leq_{out}, \sigma, MSO_{bin}[\leq_{in}]]$

Example : $\forall_x^{out} \exists_y^{out} P(\sigma(x), \sigma(y)) \rightarrow \forall_x^{out} y \geqslant_{out} x \rightarrow Q(\sigma(y), \sigma(x))$

$P, Q \in MSO[\leq_{in}]$

SATISFIABILITY Input : $\varphi \in MSO[\leq_{in}, \leq_{out}, \sigma]$ Output : $\exists G \cdot G \models \varphi$?

RESULTS

- Undecidable (even for $FO_2[\leq_{out}, S_{out}, \leq_{in}, \sigma]$)
- Decidable for $\mathcal{L}_T := FO_2[\leq_{out}, \sigma, MSO_{bin}[\leq_{in}]]$

Example : $\forall^{out}_x \exists^{out}_y P(\sigma(x), \sigma(y)) \rightarrow \forall^{out}_x y \geqslant_{out} x \rightarrow Q(\sigma(y), \sigma(x))$

$P, Q \in MSO[\leq_{in}]$

CONSEQUENCES

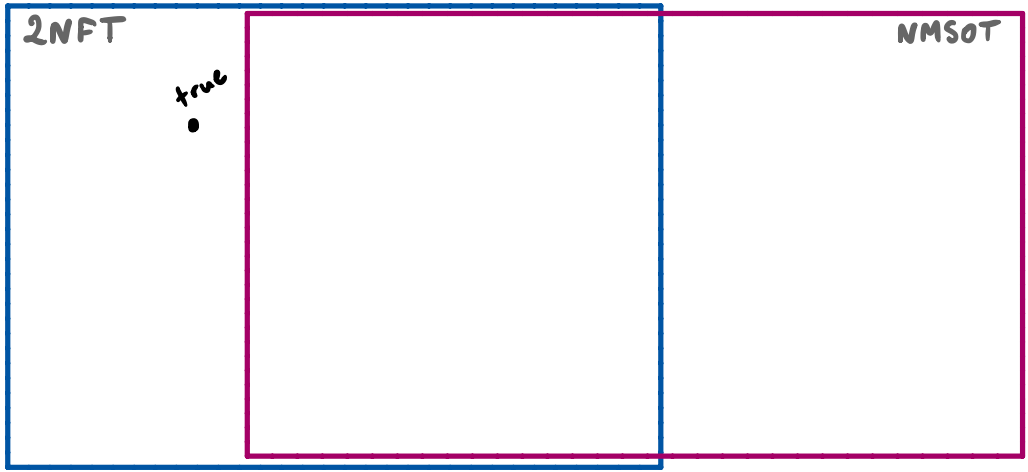
- Decidable equivalence problem for $\mathcal{L}_T$-transductions
- Decidable satisfiability for $\exists \mathcal{L}_T$

$\exists X_1 \cdots \exists X_n : \mathcal{L}_T$

$\mathcal{L}_T := FO_2[\leq_{out}, \circ, MSO_{bin}[\leq_{in}]]$

EXPRESSIVENESS OF $\mathcal{L}_T$

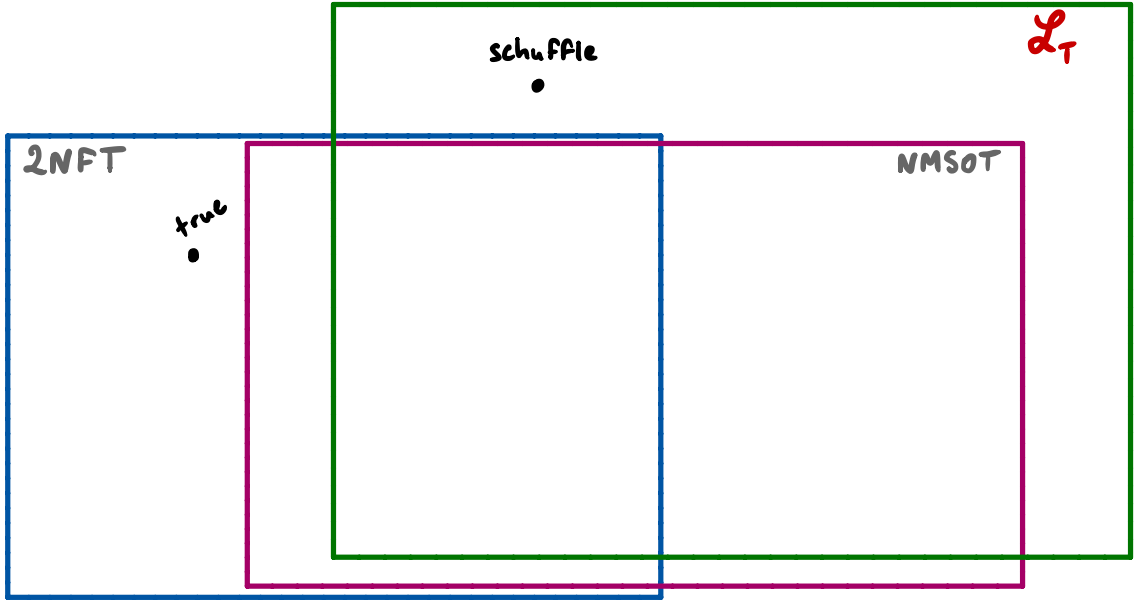
- all previous examples (including shuffle)
- $\mathcal{L}_T > 2DFT = MSOT$



$\mathcal{L}_T := FO_2[\leq_{out}, \circ, MSO_{bin}[\leq_{in}]]$

EXPRESSIVENESS OF $\mathcal{L}_T$

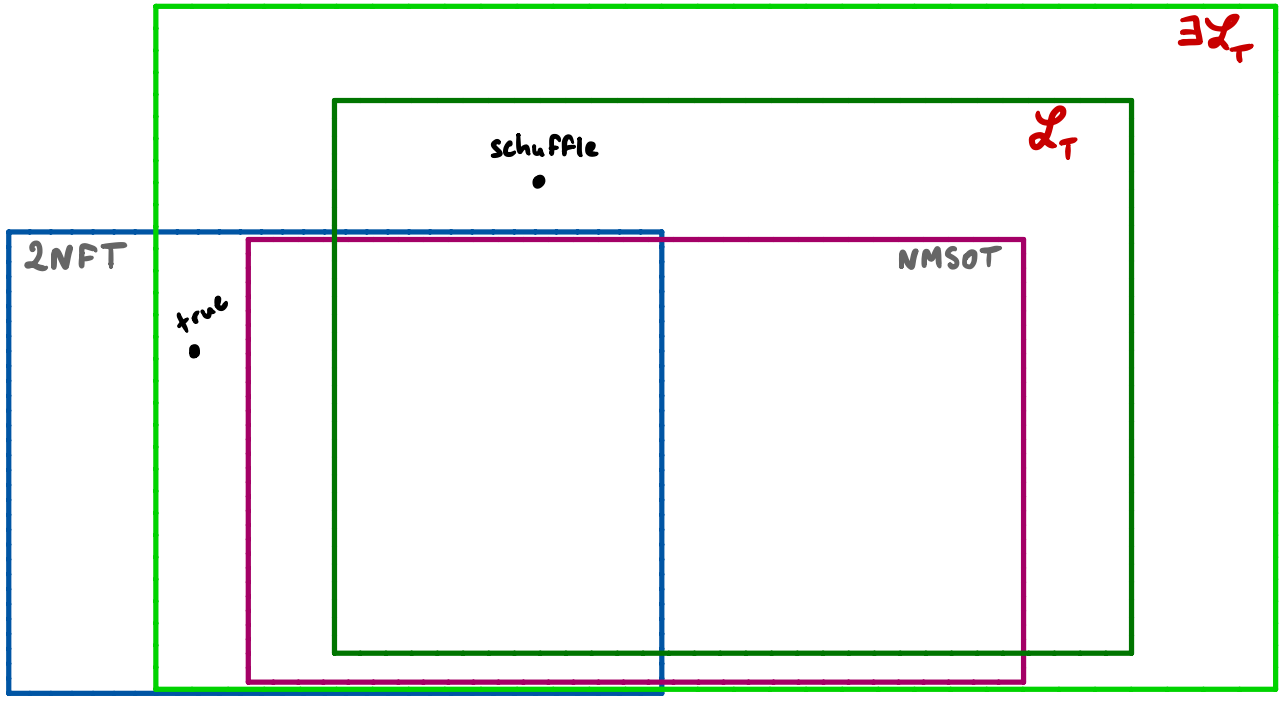
- all previous examples (including shuffle)
- $\mathcal{L}_T > 2DFT = MSOT$



$\mathcal{L}_T := FO_2[\leq_{out}, \sigma, MSO_{bin}[\leq_{in}]]$

EXPRESSIVENESS OF $\mathcal{L}_T$

- all previous examples (including shuffle)
- $\mathcal{L}_T > 2DFT = MSOT$



SYNTHESIS

THEOREM

$$\forall \varphi \in \exists\!\!\!\mathcal{L}_T \cdot \exists f : \Sigma^* \to \Sigma^* \cdot \begin{cases} 1. & f \in \text{MSOT} \\ 2. & \text{dom}\, f = \text{dom}\, \varphi \\ 3. & f \models \varphi \end{cases}$$

SYNTHESIS

THEOREM

$$\forall \varphi \in \exists \mathcal{L}_T \cdot \exists f : \Sigma^* \to \Sigma^* \cdot \begin{cases} 1. & f \in \text{MSOT} \\ 2. & \text{dom}\, f = \text{dom}\, \varphi \\ 3. & f \models \varphi \end{cases}$$

CONSEQUENCES

- New characterization of MSOT : $\text{MSOT} = \mathcal{L}_T \cap \text{functions}$
- Decidable functionality problem for $\mathcal{L}_T$

1. $\varphi : \mathcal{L}_T \xrightarrow{\text{synthesis}} \psi : \text{MSOT} \xrightarrow{\text{expressiveness}} \varphi' : \mathcal{L}_T \cap \text{functions}$
2. Test $\varphi \equiv \varphi'$

DATA WORDS (OVER Σ, D)

DEFINITION

$$(\sigma_1, d_1) \dots (\sigma_n, d_n) \in (\Sigma \times D)^*$$

infinite set of ordered data

finite alphabet

DATA WORDS (OVER Σ, D)

DEFINITION

$$(\sigma_1, d_1) \dots (\sigma_n, d_n) \in (\Sigma \times D)^*$$

$(\Sigma \times D)^*$: infinite set of ordered data

Σ: finite alphabet

DATA WORD AS FINITE STRUCTURE

finite word structure + data comparison

$\{1, \dots, n\}$: positions

$\leqslant$: linear_order

$\preccurlyeq$: preorder on positions

DATA WORDS (OVER $\Sigma, \mathbb{D}$)

DEFINITION

$$(\sigma_1, d_1) \dots (\sigma_n, d_n) \in (\Sigma \times \mathbb{D})^*$$

infinite set of ordered data

finite alphabet

DATA WORD AS FINITE STRUCTURE

finite word structure + data comparison

$\{1, \dots, n\}$: positions

$\leq$: linear_order

$\preccurlyeq$: preorder on positions

LOGICS FOR DATA WORDS

$FO_2[\leq, S_p, \sim_d]$: decidable

$FO_2[\leq, S_\downarrow, \preccurlyeq]$: decidable

$FO_2[\leq, S_p, \preccurlyeq]$: undecidable [Schwentick, Zeume, '10]

Bojańczyk
Muscholl
Schwentick '06
Segoufin
David

TYPED DATA WORDS (over $\Sigma, \mathbb{D}, T$)

DEFINITION

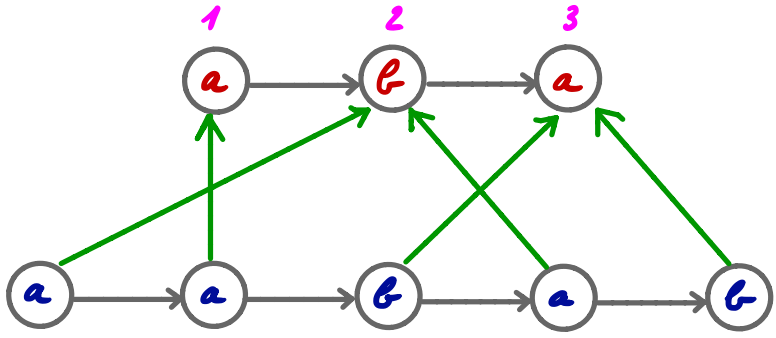
$(w, \tau : \mathbb{D} \to T)$

- w: data word
- τ: finite set of types

Closed if $data(w) = \downarrow data(w)$

$$\begin{pmatrix} d_1 \\ \epsilon_1 \\ \sigma_1 \end{pmatrix} \begin{pmatrix} d_2 \\ \epsilon_2 \\ \sigma_2 \end{pmatrix} \begin{pmatrix} d_3 \\ \epsilon_3 \\ \sigma_3 \end{pmatrix} \begin{pmatrix} d_4 \\ \epsilon_4 \\ \sigma_4 \end{pmatrix} \begin{pmatrix} d_5 \\ \epsilon_5 \\ \sigma_5 \end{pmatrix} \begin{pmatrix} d_6 \\ \epsilon_6 \\ \sigma_6 \end{pmatrix} \begin{matrix} \cdots \\ \cdots \\ \cdots \end{matrix} \begin{pmatrix} d_n \\ \epsilon_n \\ \sigma_n \end{pmatrix}$$

TYPED DATA WORDS vs TRANSDUCTIONS



data = origin !

$$\begin{pmatrix} 2 \\ b \\ a \end{pmatrix} \quad \begin{pmatrix} 1 \\ a \\ a \end{pmatrix} \quad \begin{pmatrix} 3 \\ a \\ b \end{pmatrix} \quad \begin{pmatrix} 2 \\ b \\ a \end{pmatrix} \quad \begin{pmatrix} 3 \\ a \\ b \end{pmatrix}$$

MODELS

NON-ERASING TRANSDUCTIONS WITH ORIGIN OVER Σ, Γ $\longleftrightarrow$ CLOSED TYPED DATA WORDS OVER Σ, N, Γ

MODELS

NON-ERASING TRANSDUCTIONS WITH ORIGIN OVER Σ, Γ $\quad\leftrightarrow\quad$ CLOSED TYPED DATA WORDS OVER Σ, N, Γ

LOGICS

$FO_2[\leq_{out}, \sigma, MSO_{bin}[\leq_{in}]]$ $\quad\leftrightarrow\quad$ $FO_2[\leq, MSO_{bin}[\lessdot]]$

TRANSDUCTIONS $\qquad$ DATA WORDS

MODELS

NON-ERASING TRANSDUCTIONS WITH ORIGIN OVER Σ, Γ $\quad\leftrightarrow\quad$ CLOSED TYPED DATA WORDS OVER $\Sigma, \mathbb{N}, \Gamma$

LOGICS

$FO_2[\leq_{out}, \sigma, MSO_{bin}[\leq_{in}]]$ $\quad\leftrightarrow\quad$ $FO_2[\leq, MSO_{bin}[\lessdot]]$

TRANSDUCTIONS — DATA WORDS

CONSEQUENCE

- $FO_2[\leq, MSO_{bin}[\lessdot]]$ is decidable on typed data words over $\mathbb{N}$ (closedness does not matter)
- $EMSO_2[\leq, MSO_{bin}[\lessdot]]$ is decidable too.

SUMMARY

- new logic tailored to express non-functional transductions
- decidable satisfiability / model-checking / equivalence / functionality
- MSOT synthesis
- new characterisation of MSOT
- origin-sensitive ! otherwise undecidable

SUMMARY

- new logic tailored to express non-functional transductions
- decidable satisfiability / model-checking / equivalence / functionality
- MSOT synthesis
- new characterization of MSOT
- origin-sensitive ! otherwise undecidable

FUTURE WORK

- synthesis of "memory-efficient" machines from $\mathcal{L}_T$
- extensions to trees & data word transductions

SUMMARY

- new logic tailored to express non-functional transductions
- decidable satisfiability / model-checking / equivalence / functionality
- MSOT synthesis
- new characterization of MSOT
- origin-sensitive ! otherwise undecidable

FUTURE WORK

- synthesis of "memory-efficient" machines from $\mathcal{L}_T$
- extensions to trees & data word transductions

CHALLENGE

- MSOT = 2DFT = 2RFT = SST = $\mathcal{L}_T$ functions = ...
- canonical model ?

NON-DETERMINISTIC MSO TRANSDUCERS

Use monadic second order parameters

$\varphi_{dom}(X) \equiv$ odd_cardinality (X)

$\varphi_S(x, y, X) \equiv x, y \in X \wedge \neg \exists z \in X \cdot x < z < y$

output : b a b

output : a b a