Logic-Automata Connections for Transformations

Emmanuel Filiot

Université Libre de Bruxelles

ICLA 2015

Languages of finite words

- Σ: a finite alphabet of **letters** or **symbols**
- a (finite) **word** is a finite sequence of letters over Σ
- e.g. $\Sigma = \{a, b\}$ and $w = ababba$
- ϵ is the empty word
- Σ^* is the set of finite words

A language is a subset $L \subseteq \Sigma^*$

Logic-automata connections for languages

Two formalisms for defining languages

Automata A

$w \in L(A)$ iff successful execution on w

operational

Logical sentence Φ

$w \in L(\Phi)$ iff $w \models \Phi$

declarative

Logic-automata connections for languages

Two formalisms for defining languages

Automata A

$w \in L(A)$ iff successful execution on w

operational

Logical sentence Φ

$w \in L(\Phi)$ iff $w \models \Phi$

declarative

- initiated in the 60s by Büchi, Elgot, Trakhtenbrot on finite words (over S)
  - monadic second-order logic $\equiv$ finite word automata
- then extended to other structures: infinite words (complementation, determinization), trees, ...
- and to other logics: first-order, fixpoint logics, temporal logics

Finite State Automata

- finite string acceptors over a finite alphabet Σ
- read-only input tape, left-to-right
- finite set of states

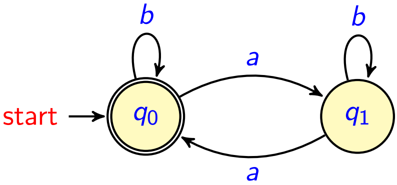
Definition (Finite State Automaton)

A finite state automaton (FA) on Σ is a tuple $A = (Q, I, F, \delta)$ where

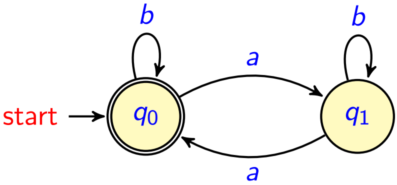
- Q is the set of states,
- $I \subseteq Q$, reps. $F \subseteq Q$ is the set of initial, resp. final, states,
- $\delta : Q \times \Sigma \to Q$ is the transition relation.

$$L(A) \;=\; \{w \in \Sigma^* \mid \text{there exists an accepting run on } w\}$$

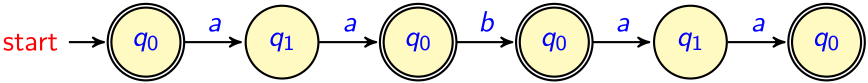
Finite State Automata – Example



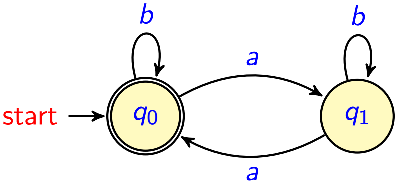
Finite State Automata – Example



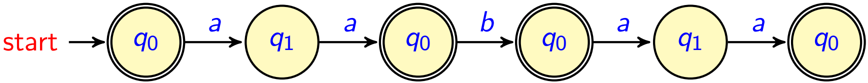
Run on *aabaa*:



Finite State Automata – Example



Run on *aabaa*:



$$L(A) = \{w \in \Sigma^* \mid w \text{ contains an even number of } a\}$$

Monadic Second-Order Logic over Finite Words

Words as logical structures

A word over some alphabet Σ can be seen as a logical structure over the signature $\{(a(.))_{a\in\Sigma}, S(.,.)\}$ (with equality $=$)

$$a \xrightarrow{S} b \xrightarrow{S} a \xrightarrow{S} a \xrightarrow{S} b \xrightarrow{S} a \xrightarrow{S} a \xrightarrow{S} b$$

MSO[S] Syntax

Second-order logic restricted to quantification over sets.

$$\phi \ ::= \ S(x,y) \mid \sigma(x) \mid x \in X \mid \forall x \cdot \phi \mid \forall X \cdot \phi \mid \neg\phi \mid \phi \wedge \phi$$

MSO[S]-definable languages

Given ϕ: MSO[S] sentence,

$$[\![\phi]\!] \ = \ \{w \in \Sigma^* \mid w \models \phi\}$$

Monadic Second-Order Logic over Finite Words

Examples

- x is the first position:

$$\mathrm{first}(x) \equiv \forall y \cdot \neg S(y, x)$$

Monadic Second-Order Logic over Finite Words

Examples

- x is the first position:

$$\mathrm{first}(x) \equiv \forall y \cdot \neg S(y, x)$$

- transitive closure of S: $x < y \ \equiv$

$$\forall X \cdot (x \in X \wedge \forall z \forall z' \cdot (z \in X \wedge z \neq y \wedge S(z, z')) \to z' \in X) \to y \in X$$

Monadic Second-Order Logic over Finite Words

Examples

- x is the first position:

$$\text{first}(x) \equiv \forall y \cdot \neg S(y, x)$$

- transitive closure of S: $x < y \equiv$

$$\forall X \cdot (x \in X \wedge \forall z \forall z' \cdot (z \in X \wedge z \neq y \wedge S(z, z')) \rightarrow z' \in X) \rightarrow y \in X$$

- MSO sentences define languages, e.g. $a^* b^*$:

$$\exists L \exists R \cdot \forall x \cdot (x \in L \vee x \in R) \wedge \forall x \in L \forall y \in R \cdot x < y \wedge a(x) \wedge b(y)$$

Monadic Second-Order Logic over Finite Words

Examples

- x is the first position:

$$\mathrm{first}(x) \equiv \forall y \cdot \neg S(y, x)$$

- transitive closure of S: $x < y \ \equiv$

$$\forall X \cdot (x \in X \wedge \forall z \forall z' \cdot (z \in X \wedge z \neq y \wedge S(z, z')) \rightarrow z' \in X) \rightarrow y \in X$$

- MSO sentences define languages, e.g. $a^* b^*$:

$$\exists L \exists R \cdot \forall x \cdot (x \in L \vee x \in R) \wedge \forall x \in L \forall y \in R \cdot x < y \wedge a(x) \wedge b(y)$$

- even number of a:

$$\exists X \cdot (\forall x \cdot a(x) \leftrightarrow x \in X) \wedge even(X)$$

Monadic Second-Order Logic over Finite Words

Examples

- x is the first position:

$$\mathsf{first}(x) \equiv \forall y \cdot \neg S(y, x)$$

- transitive closure of S: $x < y \ \equiv$

$$\forall X \cdot (x \in X \wedge \forall z \forall z' \cdot (z \in X \wedge z \neq y \wedge S(z, z')) \to z' \in X) \to y \in X$$

- MSO sentences define languages, e.g. $a^* b^*$:

$$\exists L \exists R \cdot \forall x \cdot (x \in L \vee x \in R) \wedge \forall x \in L \forall y \in R \cdot x < y \wedge a(x) \wedge b(y)$$

- even number of a:

$$\exists X \cdot (\forall x \cdot a(x) \leftrightarrow x \in X) \wedge even(X)$$

Theorem (Büchi (60), Elgot (61) , Trakhenbrot (61))

A language $L \subseteq \Sigma^*$ is definable by a finite automata iff it is definable in $MSO[S]$.

Some Applications of Logic-Automata Connections

- decidability of validity: e.g. $MSO[S]$ over finite words

$$L(\phi) = \Sigma^* \text{ iff } L(A) = \Sigma^*$$

Some Applications of Logic-Automata Connections

- decidability of validity: e.g. $MSO[S]$ over finite words

$$L(\phi) = \Sigma^* \text{ iff } L(A) = \Sigma^*$$

- decidability of Presburger arithmetic (FO over $(\mathbb{N}, +)$)

Some Applications of Logic-Automata Connections

- decidability of validity: e.g. $MSO[S]$ over finite words

$$L(\phi) = \Sigma^* \text{ iff } L(A) = \Sigma^*$$

- decidability of Presburger arithmetic (FO over $(\mathbb{N}, +)$)
- logic as a specification language: model-checking.

Input: : T: transition system, ϕ: specification
Output: $T \models \phi$?

$$T \models \phi \Leftrightarrow L(A_T) \cap L(A_{\neg\phi}) = \varnothing$$

Some Applications of Logic-Automata Connections

- decidability of validity: e.g. $MSO[S]$ over finite words

$$L(\phi) = \Sigma^* \text{ iff } L(A) = \Sigma^*$$

- decidability of Presburger arithmetic (FO over $(\mathbb{N}, +)$)
- logic as a specification language: model-checking.

Input: : T: transition system, ϕ: specification
Output: $T \models \phi$?

$$T \models \phi \Leftrightarrow L(A_T) \cap L(A_{\neg\phi}) = \varnothing$$

- logics with good complexities (Vardi,Wolper,86)

Some Applications of Logic-Automata Connections

- decidability of validity: e.g. $MSO[S]$ over finite words

$$L(\phi) = \Sigma^* \text{ iff } L(A) = \Sigma^*$$

- decidability of Presburger arithmetic (FO over $(\mathbb{N}, +)$)
- logic as a specification language: model-checking.

Input: : T: transition system, ϕ: specification
Output: $T \models \phi$?

$$T \models \phi \Leftrightarrow L(A_T) \cap L(A_{\neg\phi}) = \varnothing$$

- logics with good complexities (Vardi,Wolper,86)
- reactive-system synthesis (Büchi-Landweber): emptiness of a parity tree automaton

A definability problem

Question

Given an $MSO[<]$ formula, is it equivalent to some $FO[<]$ formula ?

A definability problem

Question

Given an $MSO[<]$ formula, is it equivalent to some $FO[<]$ formula ?

Schützenberger, McNaughton, Papert's Theorem

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

1. L is $FO[<]$-definable
2. L is definable by a star-free regular expression
3. The syntactic monoid of L is finite and aperiodic

A definability problem

Question

Given an $MSO[<]$ formula, is it equivalent to some $FO[<]$ formula ?

Schützenberger, McNaughton, Papert's Theorem

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

1. L is $FO[<]$-definable
2. L is definable by a star-free regular expression
3. The syntactic monoid of L is finite and aperiodic

Consequence

The problem $MSO[<]$ in $FO[<]$ is decidable.

1. transform the $MSO[<]$-sentence in some automaton A
2. compute the syntactic monoid of $L(A)$
3. check its aperiodicity

From Languages to Transformations

Definition

Languages over Σ	Transformations over Σ
function from Σ^* to $\{0, 1\}$	relation $R \subseteq \Sigma^* \times \Sigma^*$
accept words	transform words

From Languages to Transformations

Definition

Languages over Σ	Transformations over Σ
function from Σ^* to $\{0, 1\}$	relation $R \subseteq \Sigma^* \times \Sigma^*$
accept words	transform words

- $dom(R) = \{w \in \Sigma^* \mid \exists w' \cdot (w, w') \in R\}$
- in this talk, we mostly consider **functions** instead of relations

Examples of Transformations

- f_{del}: delete all 'a' positions

$$abbabaa \mapsto bbb$$

- f_{rev}: reverse the input word

$$stressed \mapsto desserts$$

- f_{copy}: copy the input word twice

$$ab \mapsto abab$$

- f_{halve}: maps all inputs a^n to $a^{\lfloor \frac{n}{2} \rfloor}$.

$$a^5 \mapsto a^2$$

Questions adressed in this talk

Questions

- what are the logic-based specification languages for transformations ?
- what are the operational models for transformations ?
- is there some connection between them ?

Plan

- Motivations
- Formal models of string transformations
  - String transformations
  - MSO-transducers
  - 1way and 2way-transducers
- Logic-transducer connections for regular transformations
  - 2DFT $\rightarrow$ MSOT
  - MSOT $\rightarrow$ 2DFT
  - Applications
- Logic-transducer connections for first-order transformations
  - transducers with registers
  - transition monoid for transducers with registers
- Conclusion

An operational model for transformations

Transducers

- **transducers = automata + some output mechanism**
- in this talk:
  - finite state transducers
  - two-way finite state transducers
  - register automata, aka streaming string transducers

An operational model for transformations

Transducers

- **transducers = automata + some output mechanism**
- in this talk:
  - finite state transducers
  - two-way finite state transducers
  - register automata, aka streaming string transducers

Some applications

- language and speech processing (e.g. M. Mohri's work)
- model-checking infinite state systems[a]
- verification of string-[b] and list-[c] processing programs
- databases (XML document processing)

---

[a] A survey of regular model checking, P. Abdulla, B. Jonsson, M. Nilsson, M. Saksena. 2004

[b] see BEK, developped at Microsoft Research

[c] Alur/Cerny, POPL'11

Finite State Transducers

- read-only left-to-right input head
- write-only left-to-right output head
- finite set of states

Finite State Transducers

- read-only left-to-right input head
- write-only left-to-right output head
- finite set of states

Definition (Finite State Transducers)

A finite state transducer over Σ is a pair $T = (A, O)$ where

- $A = (Q, I, F, \delta)$ is the underlying automaton
- O is an output morphism from δ to Σ^*.

- If $t = q \xrightarrow{a} q' \in \delta$, then $O(t)$ defines its output.
- $q \xrightarrow{a|w} q'$ denotes a transition whose output is $w \in \Sigma^*$.

Finite State Transducers

- read-only left-to-right input head
- write-only left-to-right output head
- finite set of states

Definition (Finite State Transducers)

A finite state transducer over Σ is a pair $T = (A, O)$ where

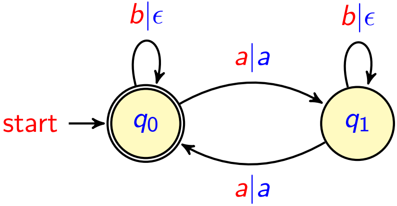
- $A = (Q, I, F, \delta)$ is the underlying automaton
- O is an output morphism from δ to Σ^*.

- If $t = q \xrightarrow{a} q' \in \delta$, then $O(t)$ defines its output.
- $q \xrightarrow{a|w} q'$ denotes a transition whose output is $w \in \Sigma^*$.

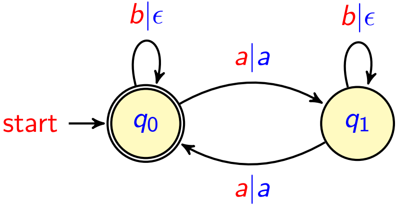
Two classes of transducers:

- DFT if A is deterministic
- NFT if A is non-deterministic.

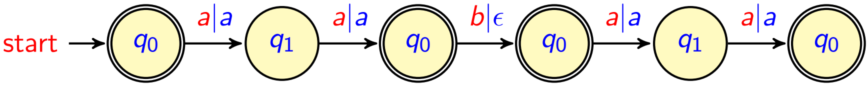
Finite State Transducers – Example 1



Finite State Transducers – Example 1

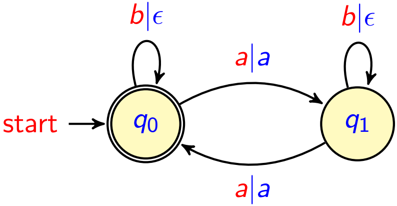


Run on *aabaa*:

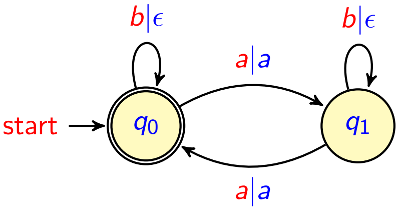


$$T(aabaa) = a.a.\epsilon.a.a = aaaa.$$

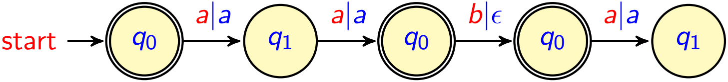
Finite State Transducers – Example 1



Finite State Transducers – Example 1

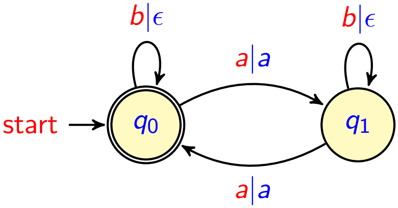


Run on *aaba*:



$T(aaba) =$ **undefined**

Finite State Transducers – Example 1



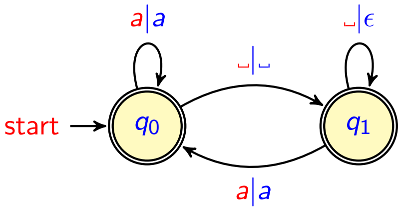
Semantics

$$dom(T) = \{w \in \Sigma^* \mid \#_a w \text{ is even}\}$$

$$R(T) = \{(w, a^{\#_a w}) \mid w \in dom(T)\}$$

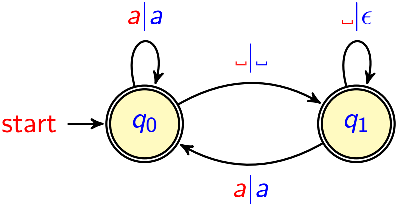
Finite State Transducers – Example 2

␣ = white space



Finite State Transducers – Example 2

␣ = white space



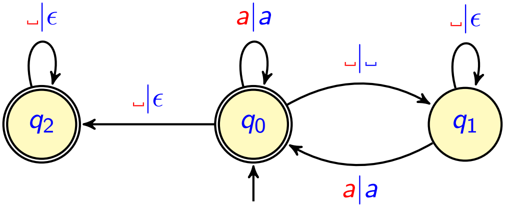
Semantics

Replace blocks of consecutive white spaces by a single white space.

$$T(␣␣aa␣␣␣a␣␣) = ␣aa␣a␣$$

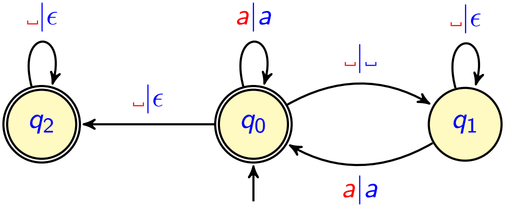
Finite State Transducers – Example 3

␣ = white space



Finite State Transducers – Example 3

␣ = white space



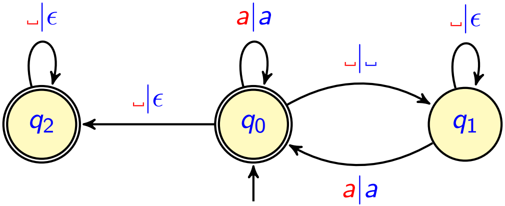
Semantics

Replace blocks of consecutive white spaces by a single white space
and
remove the last white spaces (if any).

$$T(␣␣aa␣␣␣a␣␣) = ␣aa␣a$$

Finite State Transducers – Example 3

$\llcorner\!\lrcorner$ = white space



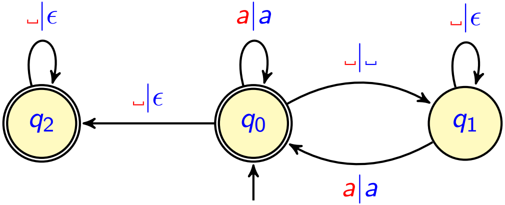
Semantics

Replace blocks of consecutive white spaces by a single white space
and
remove the last white spaces (if any).

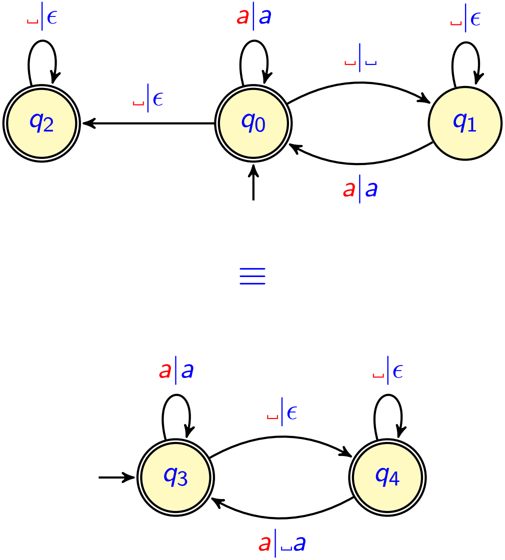
$$T(\llcorner\!\lrcorner\llcorner\!\lrcorner aa\llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner a\llcorner\!\lrcorner\llcorner\!\lrcorner) = \llcorner\!\lrcorner aa\llcorner\!\lrcorner a$$

Non-deterministic but still defines a function: functional NFT

Is non-determinism needed ?



Is non-determinism needed ?



Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

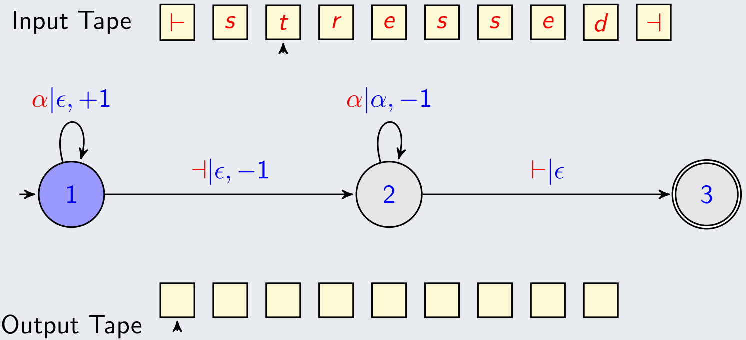
- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

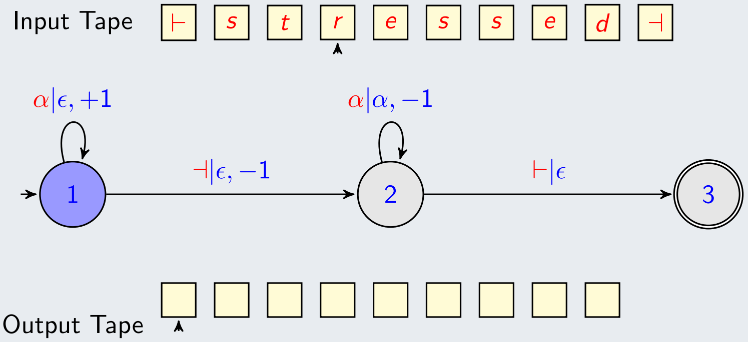


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

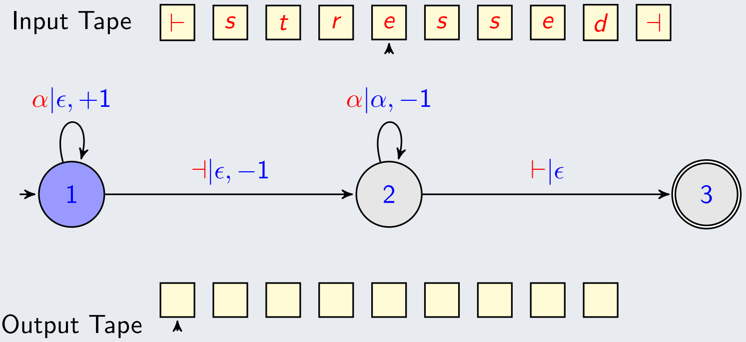


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

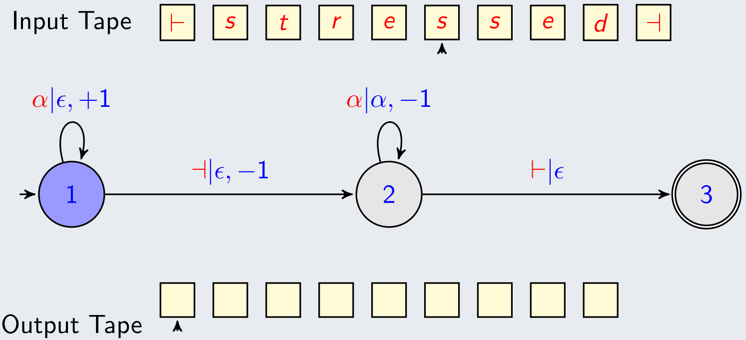


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

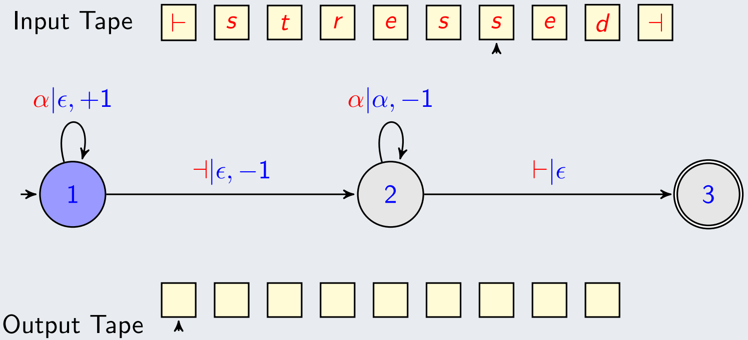


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

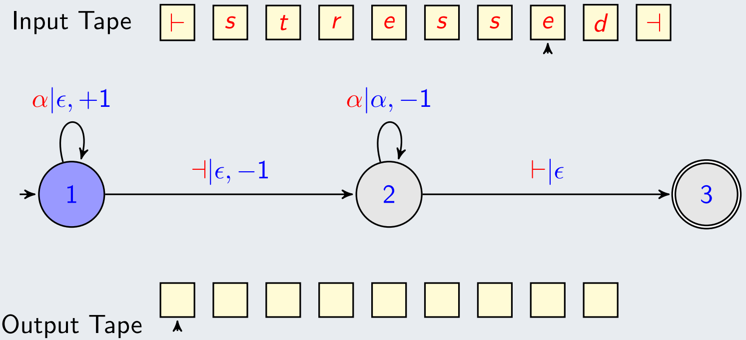


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

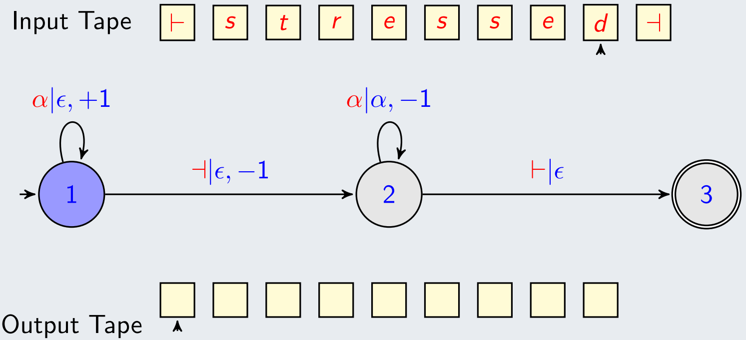


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

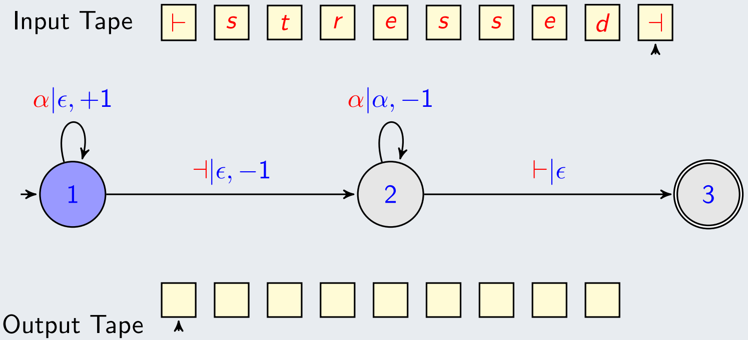


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

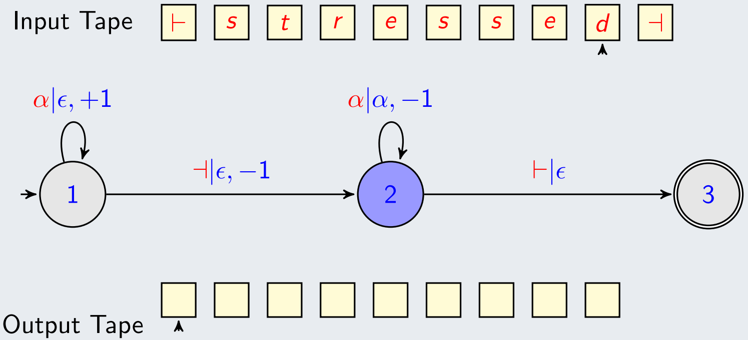


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

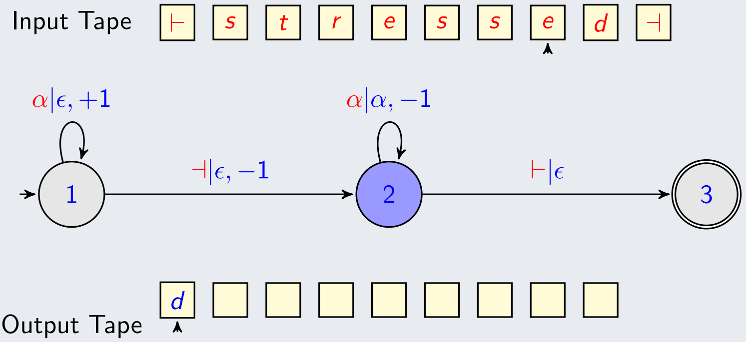


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

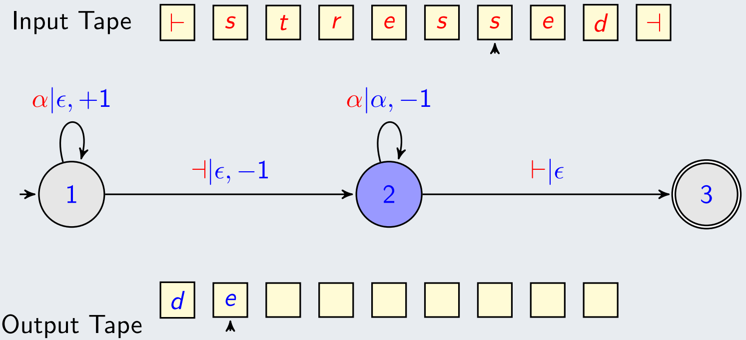


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

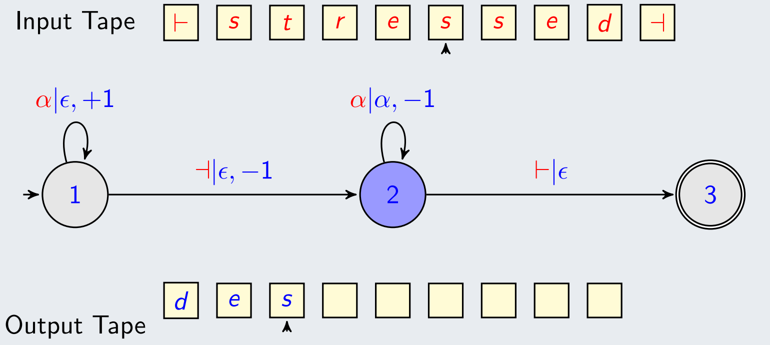


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

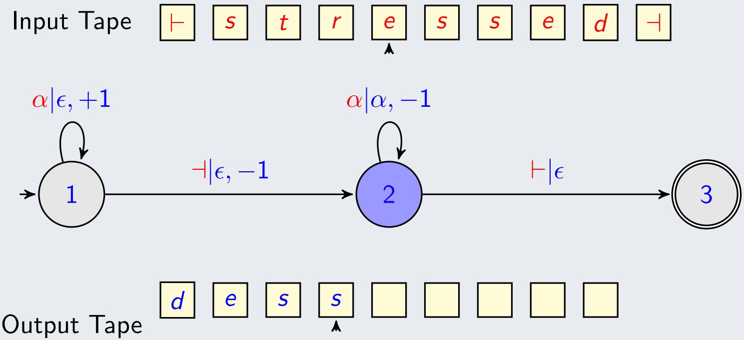


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

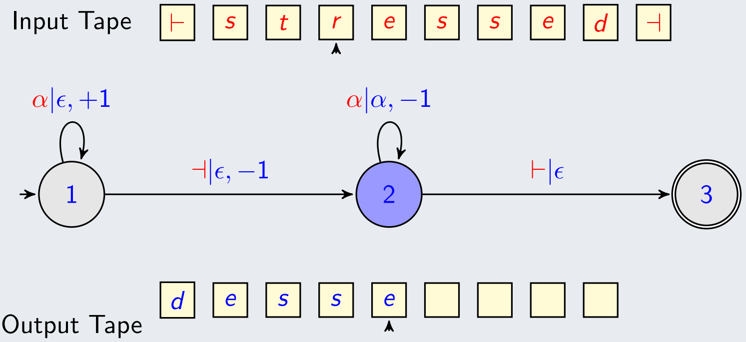


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

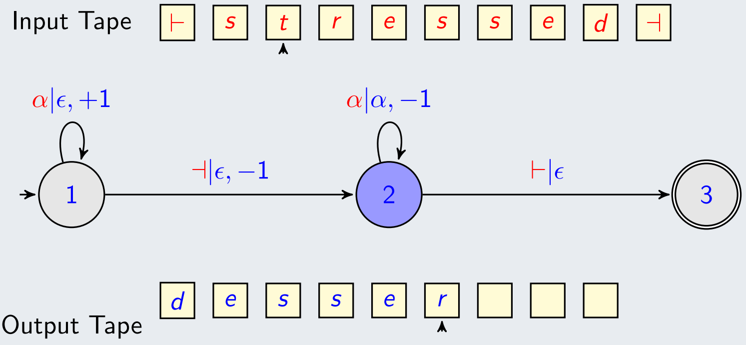


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

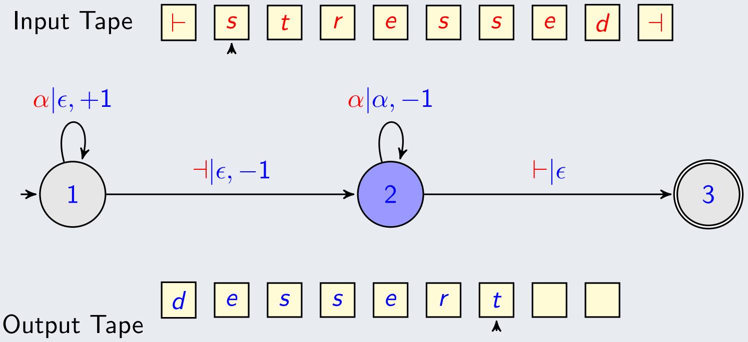


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

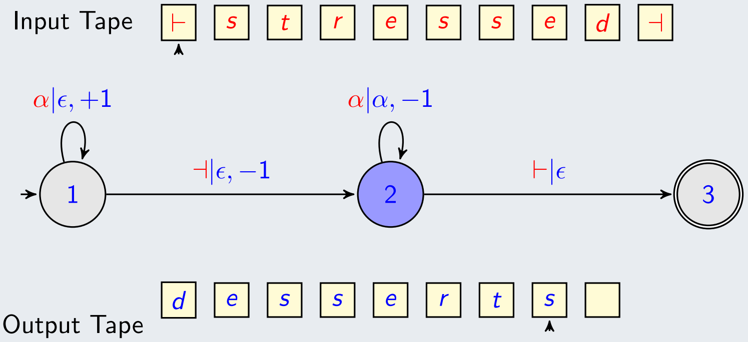


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example

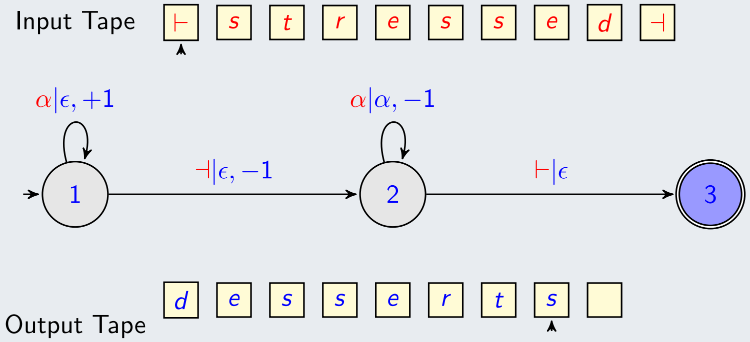


Two-way finite state transducers (2NFT)

Two-way finite state automata with outputs

- **two-way** read-only input head
- write-only left-to-right output head
- finite set of states

Example



Main Properties of Finite State Transducers

Closure under composition

- NFT are closed under composition:

$$\forall T_1, T_2 : NFT,\ \exists T : NFT,\ R(T) = R(T_1) \circ R(T_2)$$

- 2NFT are closed under composition (Chytil Jakl 77)

Main Properties of Finite State Transducers

Closure under composition

- NFT are closed under composition:

$$\forall T_1, T_2 : NFT,\ \exists T : NFT,\ R(T) = R(T_1) \circ R(T_2)$$

- 2NFT are closed under composition (Chytil Jakl 77)

Equivalence Problem

- Given T_1, T_2, does $R(T_1) = R(T_2)$ hold ?
- undecidable for NFT
- decidable for NFT and 2NFT[a] that define **functions**
- functionality is decidable for both models (in PTIME for NFT, – Gurari, Ibarra 83 –)
- decidability results extended to k-valued NFT by Culik and Karhumaki (86) and Weber (88)

[a] Culik, Karhumaki 87

(Courcelle) MSO Transformations

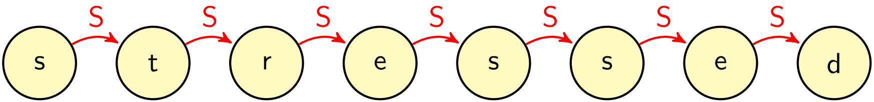
"interpreting the output structure in the input structure"

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

(Courcelle) MSO Transformations

"interpreting the output structure in the input structure"

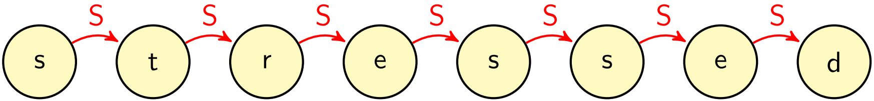
- output predicates defined by MSO[S] formulas interpreted over the input structure



(Courcelle) MSO Transformations

"interpreting the output structure in the input structure"

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



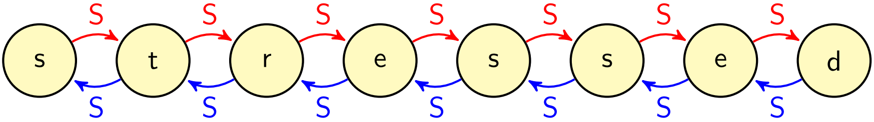
$$\phi_S(x, y) \equiv S(y, x)$$

$$\phi_{lab_a}(x) \equiv lab_a(x)$$

(Courcelle) MSO Transformations

"interpreting the output structure in the input structure"

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



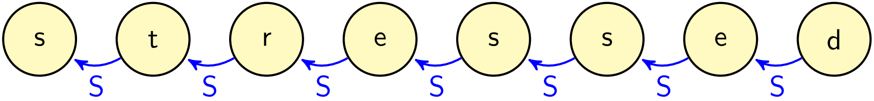
$$\phi_S(x, y) \equiv S(y, x)$$

$$\phi_{lab_a}(x) \equiv lab_a(x)$$

(Courcelle) MSO Transformations

"interpreting the output structure in the input structure"

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



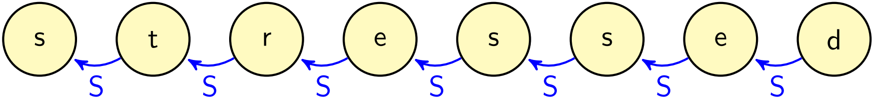
$$\phi_S(x, y) \equiv S(y, x)$$

$$\phi_{lab_a}(x) \equiv lab_a(x)$$

(Courcelle) MSO Transformations

"interpreting the output structure in the input structure"

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



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

$$\phi_{lab_a}(x) \equiv lab_a(x)$$

(Courcelle) MSO Transformations

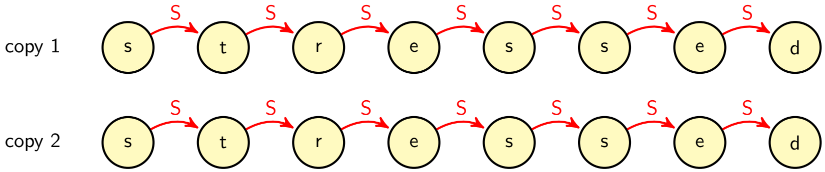
- input structure can be copied a fixed number of times:

$$w \mapsto w.f_{rev}(w)$$

(Courcelle) MSO Transformations

- input structure can be copied a fixed number of times:

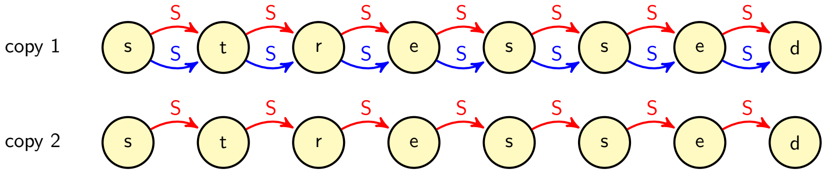
$$w \mapsto w.f_{rev}(w)$$



(Courcelle) MSO Transformations

- input structure can be copied a fixed number of times:

$$w \mapsto w.f_{rev}(w)$$



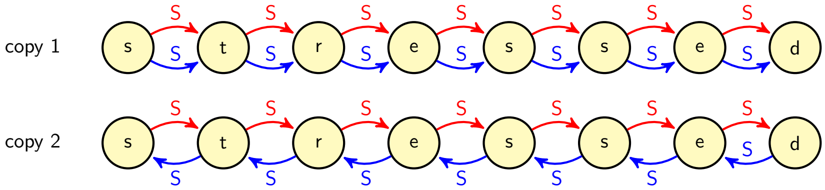
Formulas

copy 1: $\phi^1_S(x,y) \equiv S(x,y)$

(Courcelle) MSO Transformations

- input structure can be copied a fixed number of times:

$$w \mapsto w.f_{rev}(w)$$



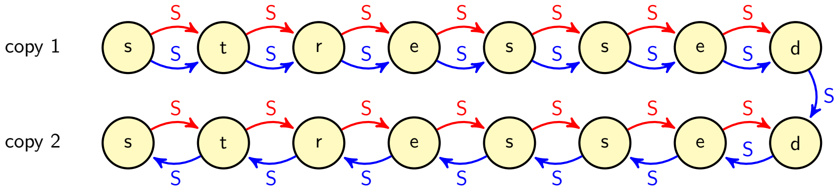
Formulas

copy 1:	$\phi^{1}_{S}(x,y)$	$\equiv$	$S(x,y)$
copy 2:	$\phi^{2}_{S}(x,y)$	$\equiv$	$S(y,x)$

(Courcelle) MSO Transformations

- input structure can be copied a fixed number of times:

$$w \mapsto w.f_{rev}(w)$$



Formulas

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

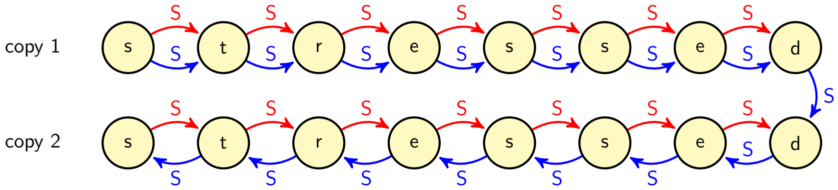
$$\textbf{copy 2:} \quad \phi_S^2(x,y) \equiv S(y,x)$$

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

(Courcelle) MSO Transformations

- input structure can be copied a fixed number of times:

$$w \mapsto w.f_{rev}(w)$$



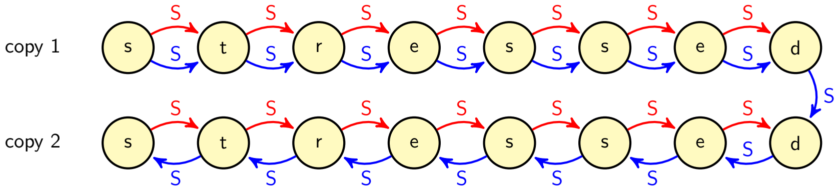
Formulas

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

(Courcelle) MSO Transformations

- input structure can be copied a fixed number of times:

$$w \mapsto w.f_{rev}(w)$$



Formulas

copy 1:	$\phi_S^1(x,y)$	$\equiv$	$S(x,y)$
copy 2:	$\phi_S^2(x,y)$	$\equiv$	$S(y,x)$
copy 1 to copy 2:	$\phi_S^{1\to 2}(x,y)$	$\equiv$	$x = y \wedge last(x)$
copy 2 to copy 1:	$\phi_S^{2\to 1}(x,y)$	$\equiv$	$\perp$
for all copies i:	$\phi^i_{lab_a}(x)$	$\equiv$	$lab_a(x)$

(Courcelle) MSO Transformations

- the domain is MSO-defined by a sentence ϕ_{dom}

Definition

An MSO transducer is defined by:

$$T = (k, \phi_{dom}, (\phi_\sigma^c(x))_{\sigma\in\Sigma, 1\leq c\leq k}, (\phi_S^{c,c'}(x,y))_{1\leq c,c'\leq k})$$

Engelfriet-Hoogeboom's Theorem

"A first Büchi theorem for transformations"

Theorem (Engelfriet,Hoogeboom,98)

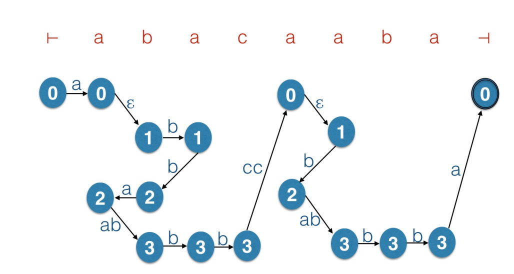
Let $f : \Sigma^* \to \Sigma^*$ be a partial function. The following statements are equivalent:

1. *f is definable by a deterministic two-way finite state transducer*
2. *f is MSOT-definable*

Moreover, the encodings are **effective** in both directions.

2DFT $\Rightarrow$ MSOT: Proof idea

Run of a 2DFT



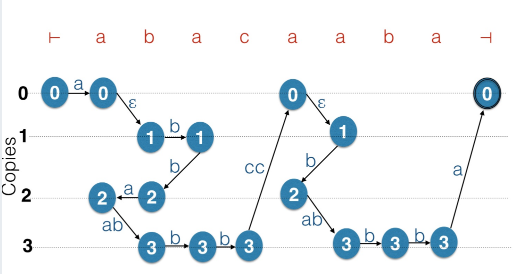
Proof Idea

1. encode the transformation from words to run graphs: T_1
   - output signature of T_1 is edge-labelled: $\{S_w(x,y) \mid w \in \Sigma^*\}$
2. transform an edge-labelled graph into a node-labelled graph over Σ: T_2
3. $T = T_2 \circ T_1$ (MSOT are closed under composition)

We focus on step 1.

2DFT $\Rightarrow$ MSOT: Proof idea

Run of a 2DFT

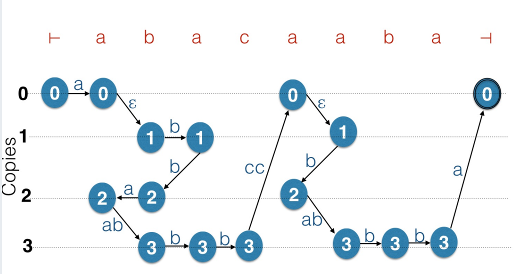


Words to run graphs: $T_1 =< k, \phi_{dom}, \phi^{c,c'}_{S_w}(x, y) >, \ 1 \leq c, c' \leq k$

- copies = states (assumed to range over $\mathbb{N}$)

2DFT $\Rightarrow$ MSOT: Proof idea

Run of a 2DFT

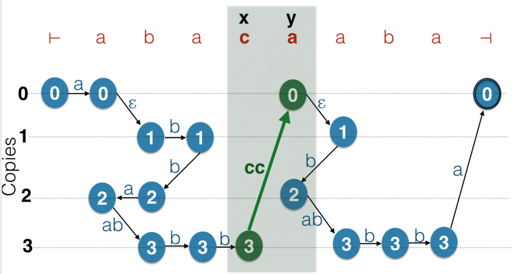


Words to run graphs: $T_1 =< k, \phi_{dom}, \phi^{c,c'}_{S_w}(x,y) >, \ 1 \leq c, c' \leq k$

- copies = states (assumed to range over $\mathbb{N}$)
- *dom*(*T*) is regular, hence *MSO*[*S*]-definable by Büchi's Theorem

2DFT $\Rightarrow$ MSOT: Proof idea

Run of a 2DFT



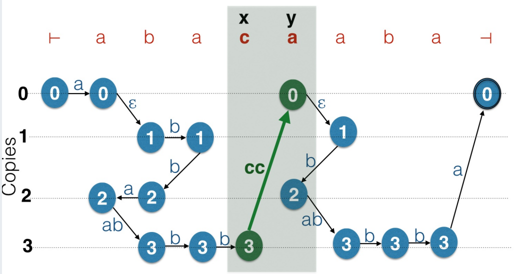
Words to run graphs: $T_1 =< k, \phi_{dom}, \phi_{S_w}^{c,c'}(x,y) >,\ 1 \leq c, c' \leq k$

- copies = states (assumed to range over $\mathbb{N}$)
- $dom(T)$ is regular, hence $MSO[S]$-definable by Büchi's Theorem
- The existence of an atomic move from state q to state p from x to y, that produces a word w, is $MSO[S]$-definable by

$$\phi_{S_w}^{q,p}(x,y) \equiv \bigvee_{(q,a,w,p,d)\in\Delta} lab_a(x) \wedge \mathrm{Run}_q(x) \wedge (d{=}1 \rightarrow S(x,y)) \wedge (d{=}{-1} \rightarrow S(y,x))$$

2DFT $\Rightarrow$ MSOT: Proof idea

Run of a 2DFT



Words to run graphs: $T_1 =< k, \phi_{dom}, \phi^{c,c'}_{S_w}(x,y) >,\ 1 \leq c, c' \leq k$

- copies = states (assumed to range over $\mathbb{N}$)
- $dom(T)$ is regular, hence $MSO[S]$-definable by Büchi's Theorem
- The existence of an atomic move from state q to state p from x to y, that produces a word w, is $MSO[S]$-definable by

$$\phi^{q,p}_{S_w}(x,y) \equiv \bigvee_{(q,a,w,p,d)\in\Delta} lab_a(x) \wedge \mathrm{Run}_q(x) \wedge (d{=}1 \rightarrow S(x,y)) \wedge (d{=}{-1} \rightarrow S(y,x))$$

- Existence of $\mathrm{Run}_q(x)$: consequence of Büchi's Theorem

MSOT⇒2DFT

Engelfriet-Hoogeboom's proof

Several intermediate models

$\boxed{\text{MSOT}} \rightarrow \boxed{2DFT^{MSO}} \rightarrow \boxed{2DFT^{RLA}} \rightarrow \boxed{\text{2DFT}}$

2*DFT* with MSO tests $\phi(x)$

and MSO jumps $\psi(x, y)$

2*DFT* with regular look-around

$$q \xrightarrow{a|w,m\in\{+1,-1\},L,R\in REG(\Sigma)} p$$

MSOT⇒2DFT

Engelfriet-Hoogeboom's proof

Several intermediate models

$\boxed{\text{MSOT}} \rightarrow \boxed{2DFT^{MSO}} \rightarrow \boxed{2DFT^{RLA}} \rightarrow \boxed{\text{2DFT}}$

$2DFT$ with MSO tests $\phi(x)$ and MSO jumps $\psi(x, y)$

$2DFT$ with regular look-around

$q \xrightarrow{a|w, m \in \{+1,-1\}, L, R \in REG(\Sigma)} p$

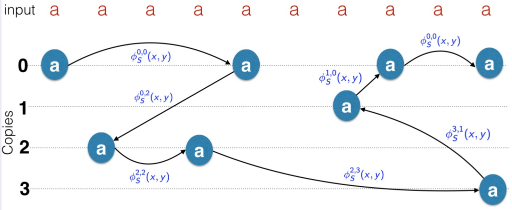
Crucial steps

- show how to simulate MSO tests by regular look-around
- show how to go from MSO jumps to walks (+1/-1 moves)
- remove look-around (uses closure under composition of 2DFT)

MSOT⇒2DFT+MSO jumps

Assumption: unary alphabet.

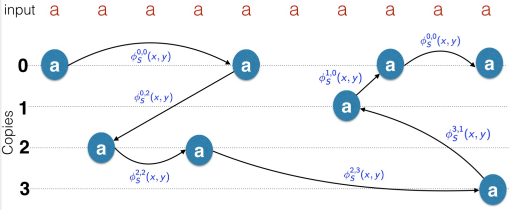
MSOT graph



MSOT⇒2DFT+MSO jumps

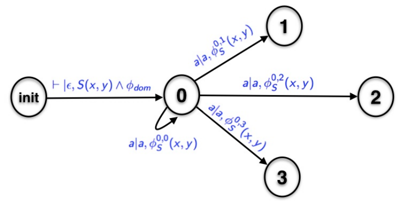
Assumption: unary alphabet.

MSOT graph



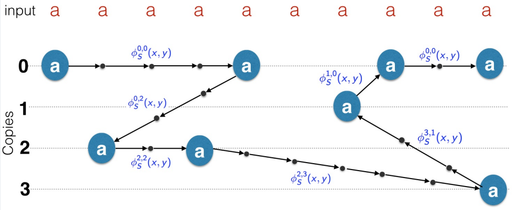
$\Downarrow$

2DFT with MSO jumps



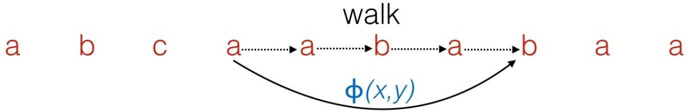
From jumps to walks

Walks



From jumps to walks

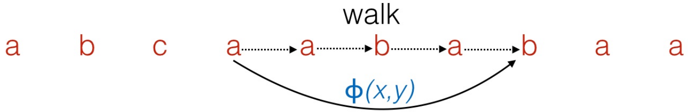
- Let $\phi(x, y)$ an MSO[S] "jump", i.e. defining a function, and such that $\phi(x, y) \Rightarrow x < y$ (other case solved similarly).



- **Goal**: replace that jump by a walk.
- **Difficulty**: the automaton starts from x, move forward, and has to determine the y position

From jumps to walks

- Let $\phi(x,y)$ an MSO[S] "jump", i.e. defining a function, and such that $\phi(x,y) \Rightarrow x < y$ (other case solved similarly).



- **Goal**: replace that jump by a walk.
- **Difficulty**: the automaton starts from x, move forward, and has to determine the y position

Idea of the construction

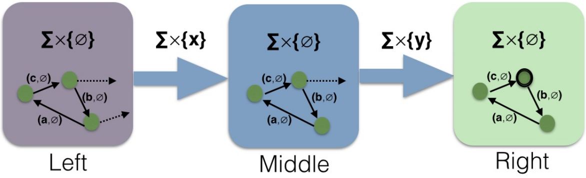
- construct a formula $\overrightarrow{\phi(x,y)}$ such that $[\![\overrightarrow{\phi(x,y)}]\!] \subseteq (\Sigma \times 2^{\{x,y\}})^*$:

$$[\![\overrightarrow{\phi(x,y)}]\!] \quad = \quad \{\overset{\varnothing}{\sigma_1} \dots \overset{\varnothing}{\sigma_{i-1}} \overset{\{x\}}{\sigma_i} \overset{\varnothing}{\sigma_{i+1}} \dots \overset{\varnothing}{\sigma_{j-1}} \overset{\{y\}}{\sigma_j} \overset{\varnothing}{\sigma_{j+1}} \dots \overset{\varnothing}{\sigma_n} \mid \sigma_1 \dots \sigma_n \models \phi(i,j)\}$$

- by Büchi's theorem, this language is regular, thus definable by a DFA $\mathcal{A}$

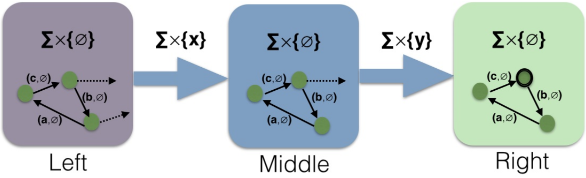
From jumps to walks

- $[\![\overrightarrow{\phi(x,y)}]\!] = \{\overset{\emptyset}{\sigma_1} \dots \overset{\emptyset}{\sigma_{i-1}} \overset{\{x\}}{\sigma_i} \overset{\emptyset}{\sigma_{i+1}} \dots \overset{\emptyset}{\sigma_{j-1}} \overset{\{y\}}{\sigma_j} \overset{\emptyset}{\sigma_{j+1}} \dots \overset{\emptyset}{\sigma_n} \mid \sigma_1 \dots \sigma_n \models \phi(i,j)\}$
- $\mathcal{A}$ can be assumed to have the following form:

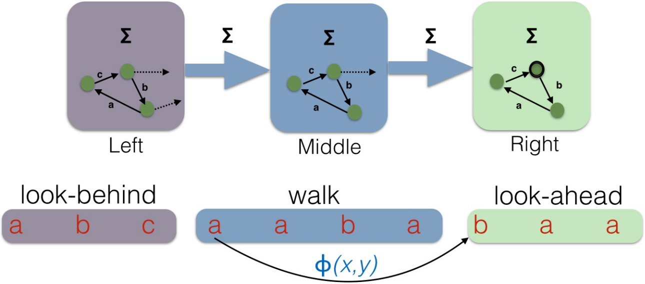


From jumps to walks

- $[\![\overrightarrow{\phi(x,y)}]\!] = \{\overset{\varnothing}{\sigma_1}\dots\overset{\varnothing}{\sigma_{i-1}}\overset{\{x\}}{\sigma_i}\overset{\varnothing}{\sigma_{i+1}}\dots\overset{\varnothing}{\sigma_{j-1}}\overset{\{y\}}{\sigma_j}\overset{\varnothing}{\sigma_{j+1}}\dots\overset{\varnothing}{\sigma_n} \mid \sigma_1\dots\sigma_n \models \phi(i,j)\}$
- $\mathcal{A}$ can be assumed to have the following form:



- project $\mathcal{A}$ on Σ, and use look-around
- first time $\mathcal{A}$ enters the right part, then current position $= y$.



2DFT+LA $\Rightarrow$ 2DFT

T : 2DFT + LA

Main idea

1. first run a 2DFT T_{la} that computes all the look-around information
   - it labels the input word with states of the look-around automata

2DFT+LA $\Rightarrow$ 2DFT

T : 2DFT + LA

Main idea

1. first run a 2DFT T_{la} that computes all the look-around information
   - it labels the input word with states of the look-around automata
2. then transform T into a 2DFT (without look-around T') that simulates T and run over tagged words

2DFT+LA $\Rightarrow$ 2DFT

T : 2DFT + LA

Main idea

1. first run a 2DFT T_{la} that computes all the look-around information
   - it labels the input word with states of the look-around automata
2. then transform T into a 2DFT (without look-around T') that simulates T and run over tagged words
3. compose T_{la} and T: it yields a 2DFT (Chytl, Jakl, 77)

Two Applications

Emptiness

- Given an MSOT ϕ, does $R(\phi) = \varnothing$ hold ?
  1. Transform ϕ into a 2DFT T
  2. Check whether $dom(T) = \varnothing$

Two Applications

Emptiness

- Given an MSOT ϕ, does $R(\phi) = \varnothing$ hold ?
  1. Transform ϕ into a 2DFT T
  2. Check whether $dom(T) = \varnothing$

Equivalence

- Given two MSOT ϕ_1, ϕ_2, does $R(\phi_1) = R(\phi_2)$ hold ?
  1. Transform ϕ_1, ϕ_2 into 2DFT T_1, T_2
  2. Check equivalence of T_1 and T_2 (decidable Chytil, Jakl, 77).

Order-preserving MSOT

- no backward edges in the MSO graph.
- e.g. reverse is not order-preserving
- can be enforced syntactically by guarding formula $\phi_S^{c,c'}(x,y)$ by $x \leq y$.

Order-preserving MSOT

- no backward edges in the MSO graph.
- e.g. reverse is not order-preserving
- can be enforced syntactically by guarding formula $\phi_S^{c,c'}(x, y)$ by $x \leq y$.

Theorem

Let $f : \Sigma^* \to \Sigma^*$ be a partial function. The following statements are equivalent:

1. *f is definable by a one-way finite state transducer*
2. *f is order-preserving MSOT-definable*

Application to some definability problem

Order-preserving problem

Given an MSOT ϕ, is ϕ equivalent to some order-preserving MSOT ?

- **Input:** MSOT T
- **Output:** Yes iff exists an order-preserving MSOT T' s.t. $R(T) = R(T')$.

Application to some definability problem

Order-preserving problem

Given an MSOT ϕ, is ϕ equivalent to some order-preserving MSOT ?

- **Input:** MSOT T
- **Output:** Yes iff exists an order-preserving MSOT T' s.t. $R(T) = R(T')$.

Theorem (F., Gauwin, Reynier, Servais, 13)

The following problem is **decidable***:*

- **Input:** *2DFT* T
- **Output***: Yes iff exists* $T' : NFT$ *s.t.* $R(T) = R(T')$*.*

Application to some definability problem

Order-preserving problem

Given an MSOT ϕ, is ϕ equivalent to some order-preserving MSOT ?

- **Input:** MSOT T
- **Output:** Yes iff exists an order-preserving MSOT T' s.t. $R(T) = R(T')$.

Theorem (F., Gauwin, Reynier, Servais, 13)

The following problem is **decidable***:*

- **Input:** *2DFT* T
- **Output***: Yes iff exists* $T' : NFT$ *s.t.* $R(T) = R(T')$.

Corollary

The order-preserving problem is decidable for MSOT.

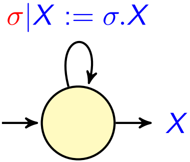
Streaming String Transducers (SST)

- one-way, deterministic model
- extend finite automata with a finite set of word variables $X, Y \ldots$
  - appending a word u: $X := Xu$
  - prepending a word: $X := uX$
  - concatenating two variables: $X := YZ$

Streaming String Transducers (SST)

- one-way, deterministic model
- extend finite automata with a finite set of word variables $X, Y \dots$
  - appending a word u: $X := Xu$
  - prepending a word: $X := uX$
  - concatenating two variables: $X := YZ$

reverse :

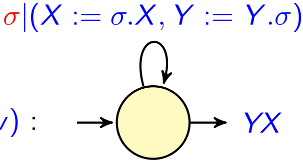


$\sigma \in \Sigma$

Streaming String Transducers (SST)

- one-way, deterministic model
- extend finite automata with a finite set of word variables $X, Y \dots$
    - appending a word u: $X := Xu$
    - prepending a word: $X := uX$
    - concatenating two variables: $X := YZ$

$w \;\mapsto\; w.f_{rev}(w)$:



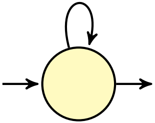
$\sigma \in \Sigma$

Streaming String Transducers (SST)

- one-way, deterministic model
- extend finite automata with a finite set of word variables $X, Y \dots$
  - appending a word u: $X := Xu$
  - prepending a word: $X := uX$
  - concatenating two variables: $X := YZ$

$\sigma | (X := \sigma.X, Y := Y.\sigma)$

$w \mapsto w.f_{rev}(w)$:



YX $\quad\quad$ $\sigma \in \Sigma$

Theorem (Alur, Cerny, 10)

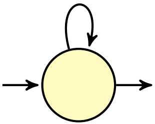
A function $f : \Sigma^* \to \Sigma^*$ is MSOT-definable iff it is definable by an SST with copyless variable update.

Streaming String Transducers (SST)

- one-way, deterministic model
- extend finite automata with a finite set of word variables $X, Y \dots$
  - appending a word u: $X := Xu$
  - prepending a word: $X := uX$
  - concatenating two variables: $X := YZ$

$\sigma | (X := \sigma.X, Y := Y.\sigma)$

$w \mapsto w.f_{rev}(w)$:



$YX \qquad \sigma \in \Sigma$

Theorem (Alur, Cerny, 10)

A function $f : \Sigma^* \to \Sigma^*$ is MSOT-definable iff it is definable by an SST with copyless variable update.

Question: What restriction to put on SST to capture FO ?

Aperiodic Finite Automata

Among several characterizations of FO languages[1], we use the following:

Theorem

A language $L \subseteq \Sigma^*$ is FO-definable iff it is definable by an aperiodic finite automaton (AFA).

Aperiodic Finite Automata

Among several characterizations of FO languages[1], we use the following:

Theorem

A language $L \subseteq \Sigma^*$ is FO-definable iff it is definable by an aperiodic finite automaton (AFA).

- AFA = finite automaton with aperiodic transition monoid $\mathcal{T}(A)$
- $\mathcal{T}(A) = \{M_w \mid w \in \Sigma^*\}$
- for any two states p, q, $M_w[p][q] = 1$ iff $p \rightsquigarrow^w q$.
- $\mathcal{T}(A)$ is aperiodic if $\exists m \geq 0$, for all $M \in \mathcal{T}(A)$, $M^m = M^{m+1}$

Aperiodic Finite Automata

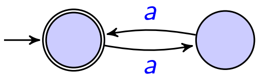
Among several characterizations of FO languages[1], we use the following:

Theorem

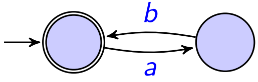
A language $L \subseteq \Sigma^*$ is FO-definable iff it is definable by an aperiodic finite automaton (AFA).

- AFA = finite automaton with aperiodic transition monoid $\mathcal{T}(A)$
- $\mathcal{T}(A) = \{M_w \mid w \in \Sigma^*\}$
- for any two states p, q, $M_w[p][q] = 1$ iff $p \rightsquigarrow^w q$.
- $\mathcal{T}(A)$ is aperiodic if $\exists m \geq 0$, for all $M \in \mathcal{T}(A)$, $M^m = M^{m+1}$
- Examples:

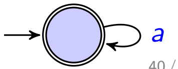
not aperiodic



aperiodic



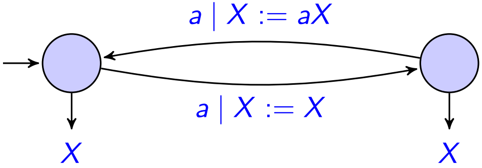
aperiodic



Towards a restriction: $f_{halve} \; : \; a^n \; \mapsto \; a^{\lfloor \frac{n}{2} \rfloor}$ again

- not FO-definable, unlike its domain
- definable by:

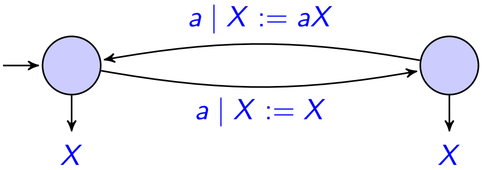
T_1 :



Towards a restriction: $f_{halve} : a^n \mapsto a^{\lfloor \frac{n}{2} \rfloor}$ again

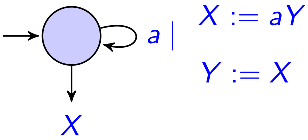
- not FO-definable, unlike its domain
- definable by:

T_1 :



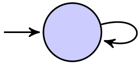
- aperiodicity of the underlying input automaton is **not sufficient**:

T_0 :



Variable flow

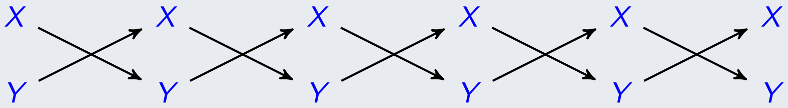
T_0 :



$a \mid \begin{array}{l} X := aY \\ Y := X \end{array}$

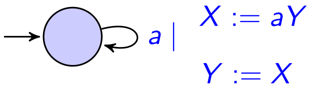
Dependency graph

input: a a a a a



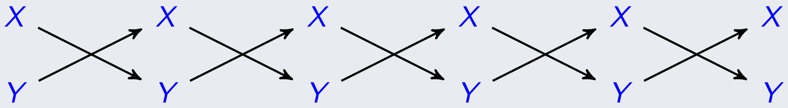
Variable flow

T_0 :



Dependency graph

input: a a a a a



$\Rightarrow$ impose aperiodicity of the variable flow !

SST Transition Monoid

- combine state flow and variable flow
- matrices indexed by pairs (q, X)
- coefficients in $\mathbb{N} \cup \{\bot\}$

SST Transition Monoid

- combine state flow and variable flow
- matrices indexed by pairs (q, X)
- coefficients in $\mathbb{N} \cup \{\perp\}$

Definition

The transition monoid of an SST is the set of $\mathbb{N} \cup \{\perp\}$-valued square matrices M_w indexed by $Q \times \mathsf{Vars}$, for all $w \in \Sigma^*$, such that

- $M_w[p, X][q, Y] = \perp$ if
  - there no run from p to q on w
- $M_w[p, X][q, Y] = n \in \mathbb{N}$ if
  - there is a run r from p to q on w
  - on this run, X "flows" n times to Y

In other words, the value of X before r appears n times in the value of Y after r.

SST Transition Monoid Examples

- if the run on aa is:

$$q_0 \xrightarrow[a]{\begin{array}{l} X := aXb \\ Y := aXa \end{array}} q_1 \xrightarrow[a]{\begin{array}{l} X := \epsilon \\ Y := YY \end{array}} q_2$$

then $M_{aa}[q_0, X][q_2, Y] = 2$ and $M_{aa}[q_0, X][q_2, Y] = 0$.

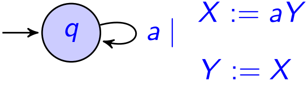
SST Transition Monoid Examples

- if the run on aa is:

$$q_0 \xrightarrow[\;]{a \;\left|\; \begin{array}{l} X := aXb \\ Y := aXa \end{array}\right.} q_1 \xrightarrow[\;]{a \;\left|\; \begin{array}{l} X := \epsilon \\ Y := YY \end{array}\right.} q_2$$

then $M_{aa}[q_0, X][q_2, Y] = 2$ and $M_{aa}[q_0, X][q_2, Y] = 0$.

- for T_0 :



$$M_{a^{2n}} = \begin{array}{c c} & \begin{array}{cc} (q,X) & (q,Y) \end{array} \\ \begin{array}{c} (q,X) \\ (q,Y) \end{array} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \text{ and } M_{a^{2n+1}} = \begin{array}{c c} & \begin{array}{cc} (q,X) & (q,Y) \end{array} \\ \begin{array}{c} (q,X) \\ (q,Y) \end{array} & \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{array}$$

Result

Theorem (F., Krishna S., Trivedi, 14)

1. *A function $f : \Sigma^* \to \Sigma^*$ is MSO-definable iff it is definable by a SST with <u>finite transition monoid</u>.*
2. *A function $f : \Sigma^* \to \Sigma^*$ is FO-definable iff it is definable by a SST with <u>aperiodic $(\perp, 0, 1)$-transition monoid</u>.*

Result

Theorem (F., Krishna S., Trivedi, 14)

1. *A function $f : \Sigma^* \to \Sigma^*$ is MSO-definable iff it is definable by a SST with finite transition monoid.*
2. *A function $f : \Sigma^* \to \Sigma^*$ is FO-definable iff it is definable by a SST with aperiodic $(\perp, 0, 1)$-transition monoid.*

Theorem (L. Dartois, O. Carton, CSL'15)

A function $f : \Sigma^* \to \Sigma^*$ is FO-definable iff it is definable by an aperiodic 2DFT.

Conclusion

Results

- transducer-logic connections for finite word functions
  - MSOT = 2DFT = SST
  - order-preserving MSOT = functional NFT
  - FOT = aperiodic SST

Conclusion

Results

- transducer-logic connections for finite word functions
  - MSOT = 2DFT = SST
  - order-preserving MSOT = functional NFT
  - FOT = aperiodic SST

Applications

- emptiness of logical-transformations
- equivalence of logical-transformations

Conclusion

Results

- transducer-logic connections for finite word functions
  - MSOT = 2DFT = SST
  - order-preserving MSOT = functional NFT
  - FOT = aperiodic SST

Applications

- emptiness of logical-transformations
- equivalence of logical-transformations

Open problems

- FOT definability problem (decidable for rational functions – N. Lothe, 2015 –)
- logics for transformations with better complexity
- model-checking techniques for word-processing programs

What else can be found in the paper ?

- FOT definability problem for a weaker semantics: transformations with origins (Bojanczyk 14)
- extensions to relations
- extension to infinite words
- extensions to trees (macro tree transducers)

What else can be found in the paper ?

- FOT definability problem for a weaker semantics: transformations with origins (Bojanczyk 14)
- extensions to relations
- extension to infinite words
- extensions to trees (macro tree transducers)

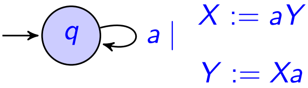
Thank You

What this result is not

Theorem

A function $f : \Sigma^* \to \Sigma^*$ is FO-definable iff it is definable by a SST with $\{0, 1, \perp\}$-valued and aperiodic transition monoid.

- It does not yield an effective characterization of FO-definable SST-transformations !
- a non-aperiodic SST can still define an FO-transformation (here the identity):

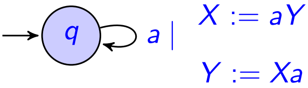


What this result is not

Theorem

A function $f : \Sigma^* \to \Sigma^*$ is FO-definable iff it is definable by a SST with $\{0, 1, \perp\}$-valued and aperiodic transition monoid.

- It does not yield an effective characterization of FO-definable SST-transformations !
- a non-aperiodic SST can still define an FO-transformation (here the identity):



- for **automata**: a language L is FO-definable iff its minimal DFA is aperiodic.