Transducer Theory and Streaming Transformations

Emmanuel Filiot

Université Libre de Bruxelles

Finite State Automata

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

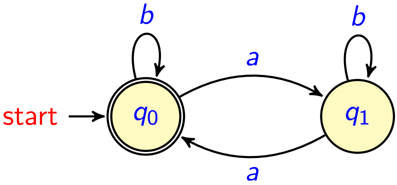
Definition (Finite State Automaton)

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

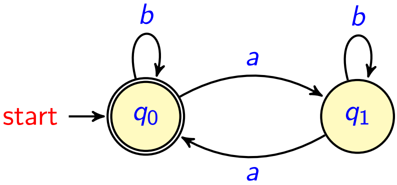
- Q is the set of states,
- $I \subseteq Q$, reps. $F \subseteq Q$ is the set of initial, resp. final, states,
- $\delta : Q \times \Sigma \to Q$ is the transition relation.

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

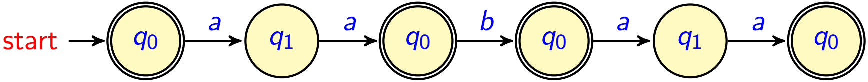
Finite State Automata – Example



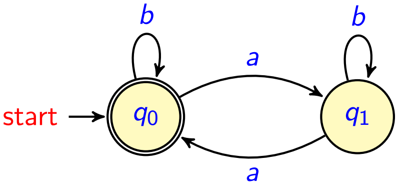
Finite State Automata – Example



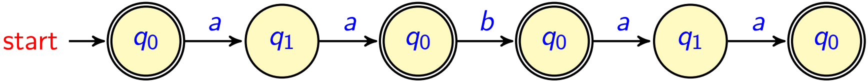
Run on *aabaa*:



Finite State Automata – Example



Run on *aabaa*:



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

Properties of FA

Expressiveness

FA = regular languages = MSO[+1] = regular expressions = ...

Properties of FA

Expressiveness

FA = regular languages = MSO[+1] = regular expressions = ...

Closure Properties

- closed under Boolean operations (union, intersection, complement).
- closed under various extensions:
  - non-determinism (NFA): $\delta \subseteq Q \times \Sigma \times Q$
  - two-way input head (2NFA): $\delta \subseteq Q \times \Sigma \times \{-1, 0, 1\} \times Q$
  - regular look-ahead: $\delta \subseteq Q \times \Sigma \times \mathit{Reg} \times Q$
  - alternation: $\delta : Q \times \Sigma \to B(Q)$ (Boolean formulas over Q)

Properties of FA

Expressiveness

FA = regular languages = MSO[+1] = regular expressions = ...

Closure Properties

- closed under Boolean operations (union, intersection, complement).
- closed under various extensions:
  - non-determinism (NFA): $\delta \subseteq Q \times \Sigma \times Q$
  - two-way input head (2NFA): $\delta \subseteq Q \times \Sigma \times \{-1, 0, 1\} \times Q$
  - regular look-ahead: $\delta \subseteq Q \times \Sigma \times Reg \times Q$
  - alternation: $\delta : Q \times \Sigma \to B(Q)$ (Boolean formulas over Q)

Decision Problems

Membership, emptiness, universality, inclusion, equivalence ... are decidable.

From Languages to Transductions

Let Σ and Δ be two finite alphabets.

Definition

Language on Σ	Transduction from Σ to Δ
function from Σ^* to $\{0,1\}$	relation $R \subseteq \Sigma^* \times \Delta^*$
defined by automata	defined by transducers
accept strings	transform strings

transducer $=$ <u>automaton</u> $+$ output mechanism.

Finite State Transducers

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 from Σ to Δ is a pair $T = (A, O)$ where

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

- 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 $w \in \Delta^*$.

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 from Σ to Δ is a pair $T = (A, O)$ where

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

- 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 $w \in \Delta^*$.

Two classes of transducers:

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

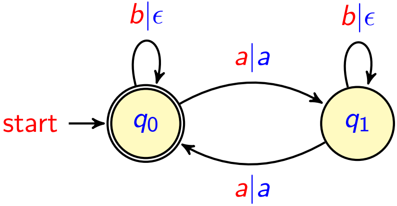
Some applications

- language and speech processing (e.g. see work by Mehryar Mohri)
- model-checking infinite state-space systems[1]
- verification of web sanitizers[2]
- string pattern matching

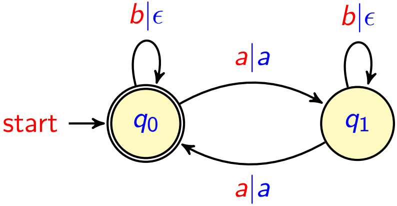
[1] A survey of regular model checking, P. Abdulla, B. Jonsson, M. Nilsson, M. Saksena. 2004

[2] see BEK, developped at Microsoft Research

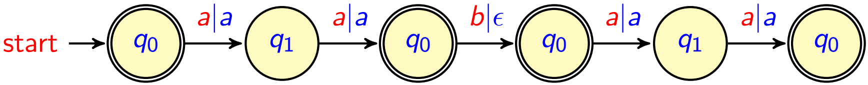
Finite State Transducers – Example 1



Finite State Transducers – Example 1

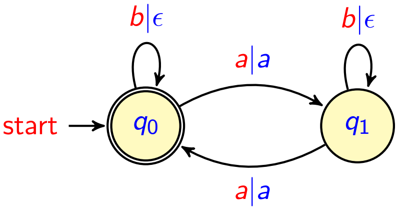


Run on $aabaa$:

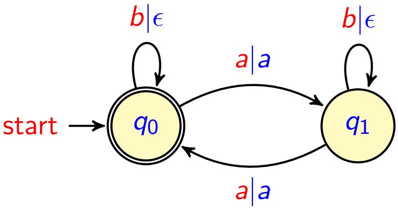


$$T(aabaa) = a.a.\epsilon.a.a = aaaa.$$

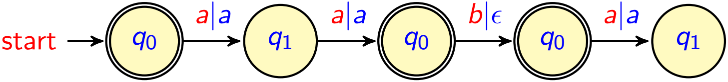
Finite State Transducers – Example 1



Finite State Transducers – Example 1

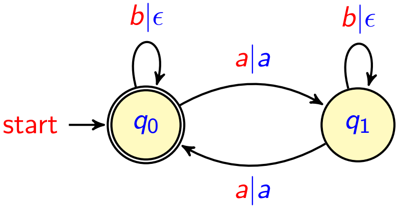


Run on *aaba*:



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

Finite State Transducers – Example 1

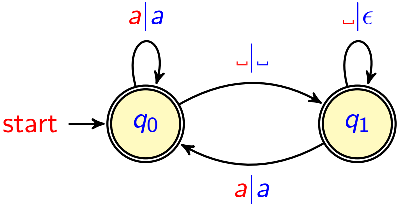


Semantics

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

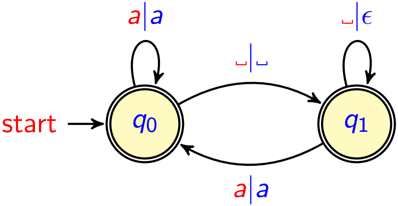
Finite State Transducers – Example 2

␣ = white space



Finite State Transducers – Example 2

␣ = white space



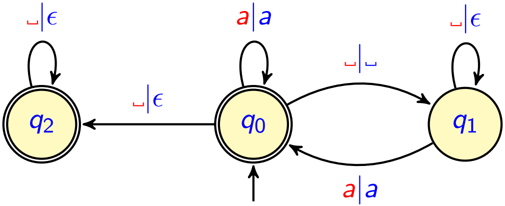
Semantics

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

$$T(\text{␣␣}aa\text{␣␣␣}a\text{␣␣}) = \text{␣}aa\text{␣}a\text{␣}$$

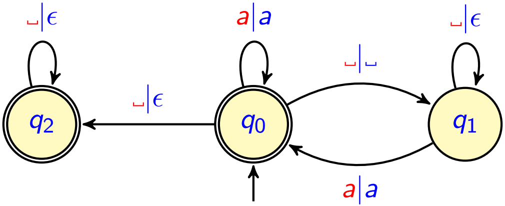
Finite State Transducers – Example 3

␣ = white space



Finite State Transducers – Example 3

$\llcorner\!\lrcorner$ = white space



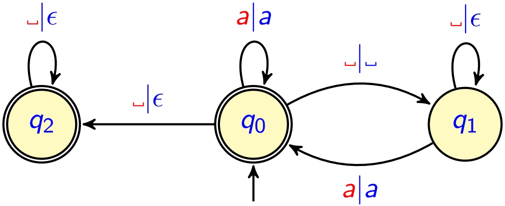
Semantics

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

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

Finite State Transducers – Example 3

$\llcorner\!\lrcorner$ = white space



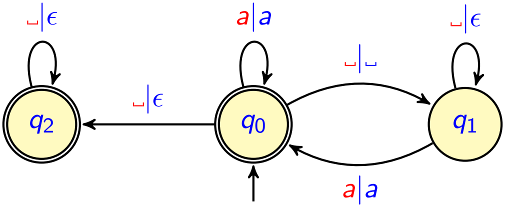
Semantics

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

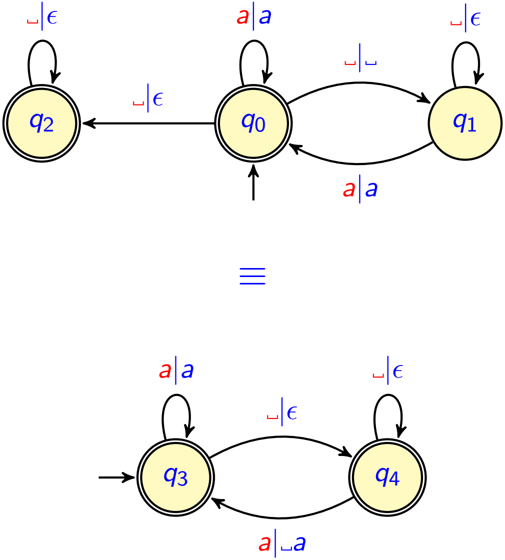
$$T(\llcorner\!\lrcorner\llcorner\!\lrcorner aa\llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner a\llcorner\!\lrcorner\llcorner\!\lrcorner) = \llcorner\!\lrcorner aa\llcorner\!\lrcorner a$$

Non-deterministic but still defines a function: functional NFT

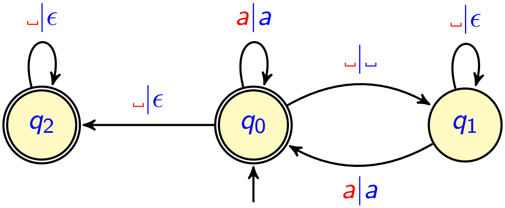
Is non-determinism needed ?



Is non-determinism needed ?

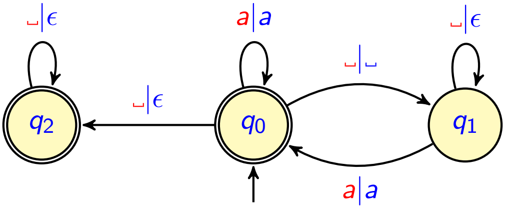


How to get a deterministic FT ?

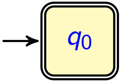


- extend automata subset construction with outputs
- output the longest common prefix

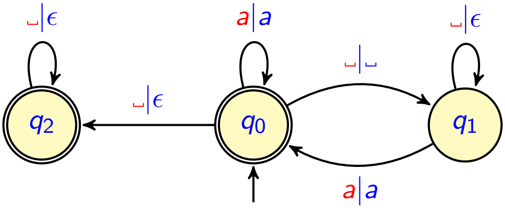
How to get a deterministic FT ?



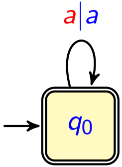
- extend automata subset construction with outputs
- output the longest common prefix



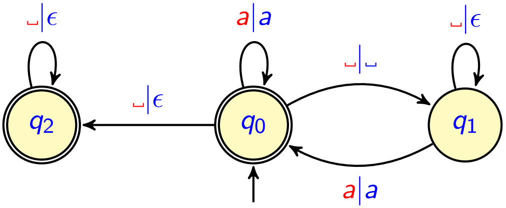
How to get a deterministic FT ?



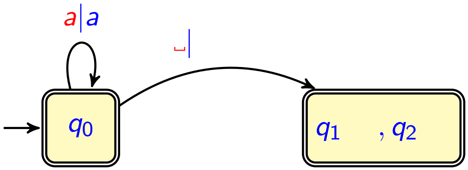
- extend automata subset construction with outputs
- output the longest common prefix



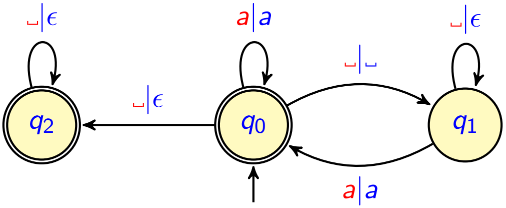
How to get a deterministic FT ?



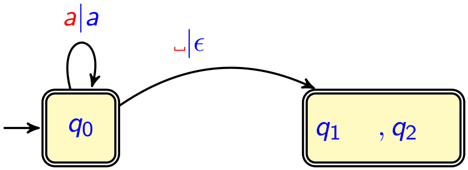
- extend automata subset construction with outputs
- output the longest common prefix



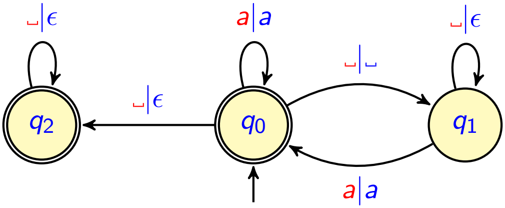
How to get a deterministic FT ?



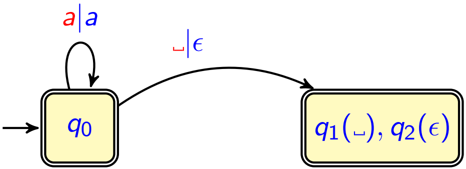
- extend automata subset construction with outputs
- output the longest common prefix



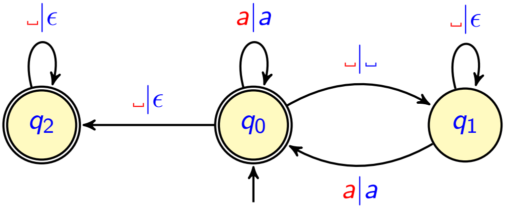
How to get a deterministic FT ?



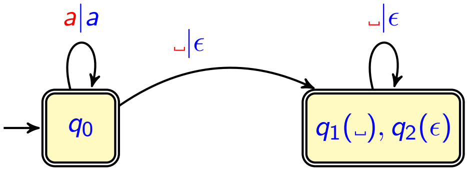
- extend automata subset construction with outputs
- output the longest common prefix



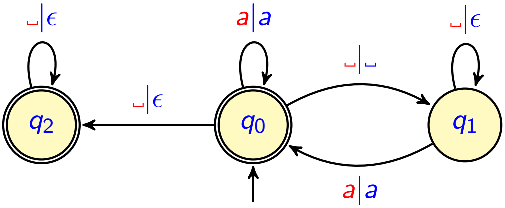
How to get a deterministic FT ?



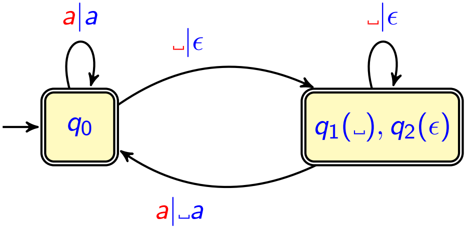
- extend automata subset construction with outputs
- output the longest common prefix



How to get a deterministic FT ?



- extend automata subset construction with outputs
- output the longest common prefix



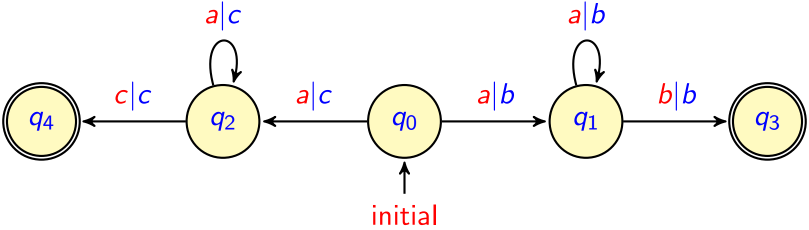
Can we always get an equivalent deterministic FT ?

Can we always get an equivalent deterministic FT ?

- not in general: DFT define functions, NFT define relations
- what about functional NFT ?

Can we always get an equivalent deterministic FT ?

- not in general: DFT define functions, NFT define relations
- what about functional NFT ?

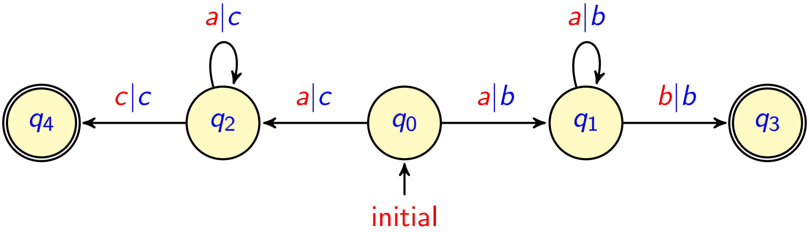


Semantics

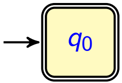
$$R(T): \begin{cases} a^n b \mapsto b^{n+1} \\ a^n c \mapsto c^{n+1} \end{cases}$$

functional but not determinizable

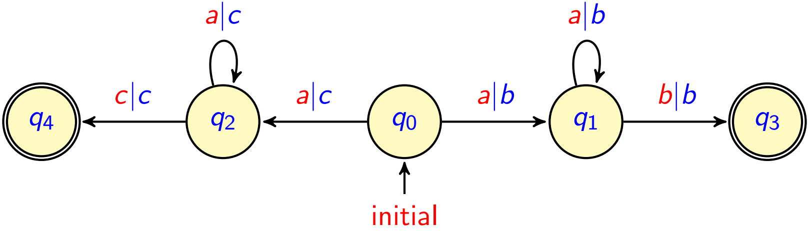
Subset construction fails ...



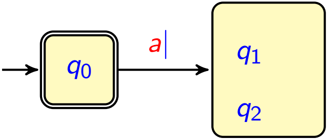
Subset construction:



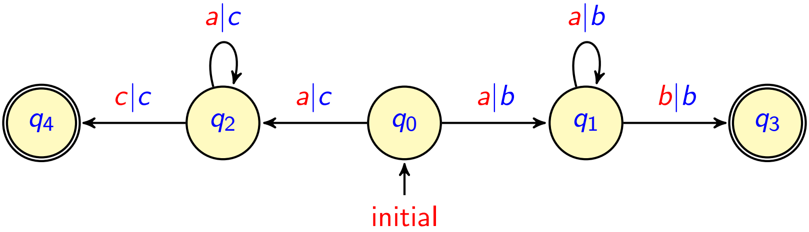
Subset construction fails ...



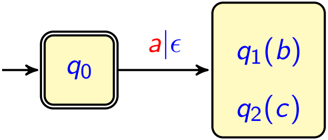
Subset construction:



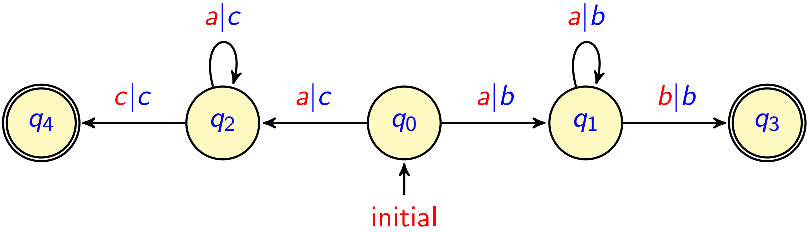
Subset construction fails ...



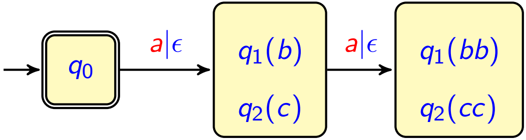
Subset construction:



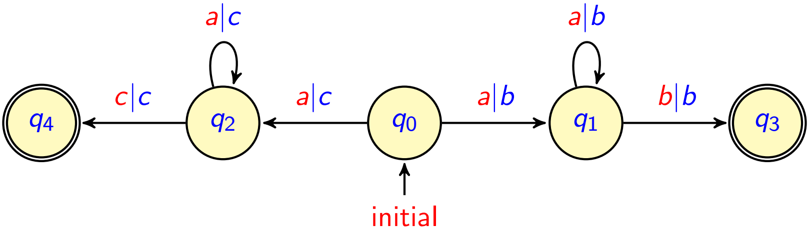
Subset construction fails ...



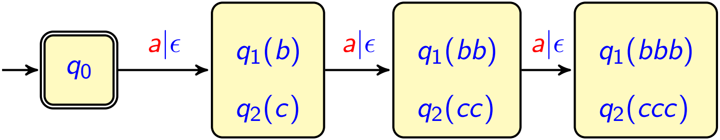
Subset construction:



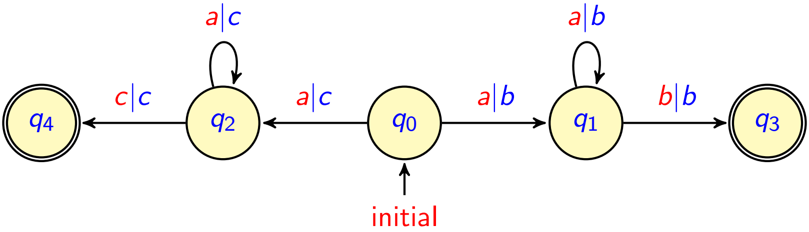
Subset construction fails ...



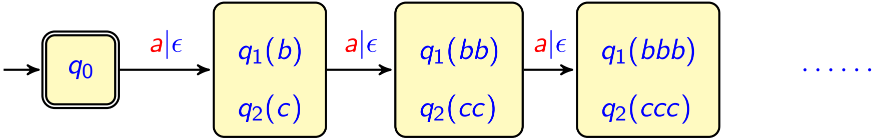
Subset construction:



Subset construction fails ...



Subset construction:



How to guarantee termination of subset construction?

LAG

$LAG(u, v) = (u', v')$ such that $u = \ell u'$, $v = \ell v'$ and $\ell = lcp(u, v)$.

E.g. $LAG(abbc, abc) = (bc, c)$.

How to guarantee termination of subset construction?

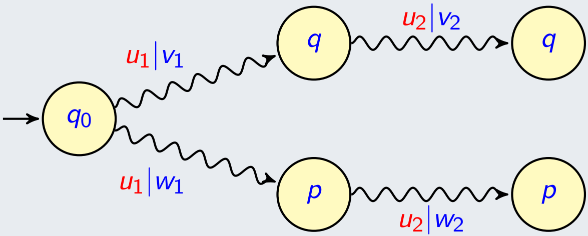
LAG

$LAG(u, v) = (u', v')$ such that $u = \ell u'$, $v = \ell v'$ and $\ell = lcp(u, v)$.

E.g. $LAG(abbc, abc) = (bc, c)$.

Lemma (Twinning Property)

Subset construction terminates **iff** *for all such situations*



it is the case that $LAG(v_1, w_1) = LAG(v_1 v_2, w_1 w_2)$.

Determinizability is decidable

Theorem (Choffrut 77, Beal Carton Prieur Sakarovitch 03)

Given a functional NFT T, the following are equivalent:

1. *it is determinizable*
2. *the twinning property holds.*

Moreover, the twinning property is decidable in PTime.

Proof.

Intuition

- If TP holds, then subset construction terminates and produces an equivalent DFT
- for the converse, uses the fact that TP is machine-independent: for all $T \equiv T'$, $T \models TP$ iff $T' \models TP$.

$\square$

Determinizability is decidable

Theorem (Choffrut 77, Beal Carton Prieur Sakarovitch 03)

Given a functional NFT T, the following are equivalent:

1. *it is determinizable*
2. *the twinning property holds.*

Moreover, the twinning property is decidable in PTime.

Proof.

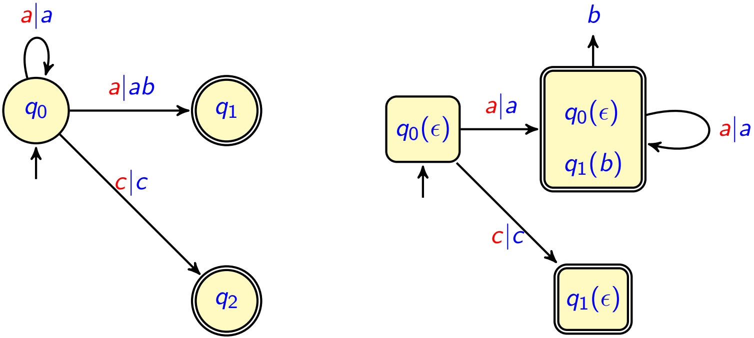
Intuition

- If TP holds, then subset construction terminates and produces an equivalent DFT
- for the converse, uses the fact that TP is machine-independent: for all $T \equiv T'$, $T \models TP$ iff $T' \models TP$.

$\square$

Almost true ...

True if ...



- subsequential transducers are deterministic but can output a string in each accepting states
- in the previous theorem: "determinizable" $\leftrightarrow$ "there exists an equivalent subsequential transducer"
- subsequential transducers $\equiv$ DFT if last string symbol is unique

Application: analysis of streaming transformations

Bounded Memory Problem

Hypothesis:

- input string is received as a (very long) **stream**
- output string is produced as a stream

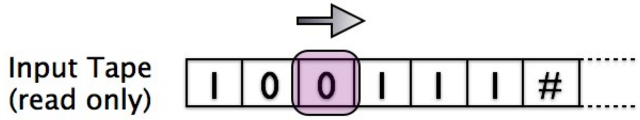
Input: a transformation defined by some functional NFT
Output: can I realize this transformation with bounded memory ?

$$\exists B \in \mathbb{N} \cdot \forall u \in dom(T)$$

$T(u)$ can be computed with B-bounded memory ?

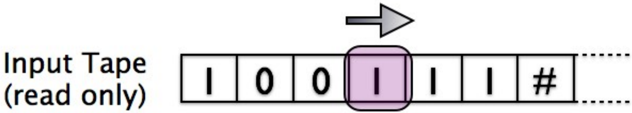
Streaming Model

Deterministic Turing Transducer



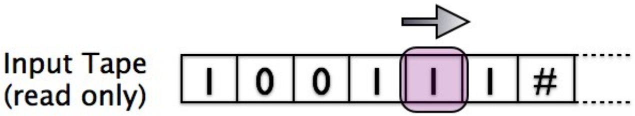
Streaming Model

Deterministic Turing Transducer



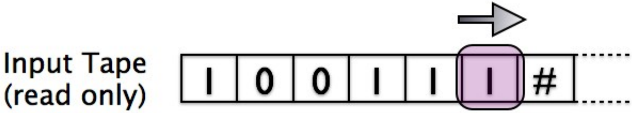
Streaming Model

Deterministic Turing Transducer



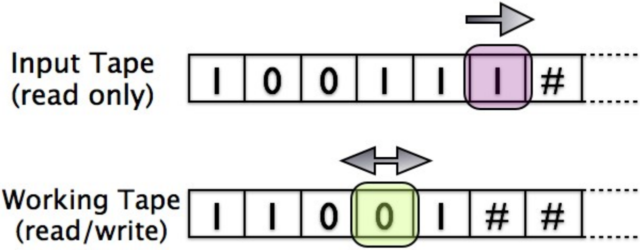
Streaming Model

Deterministic Turing Transducer



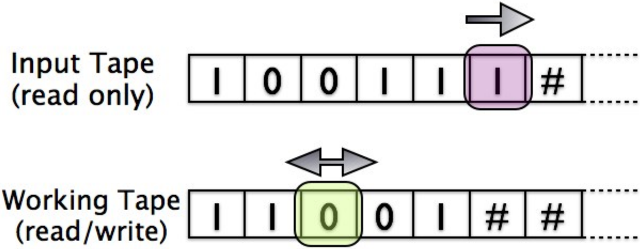
Streaming Model

Deterministic Turing Transducer



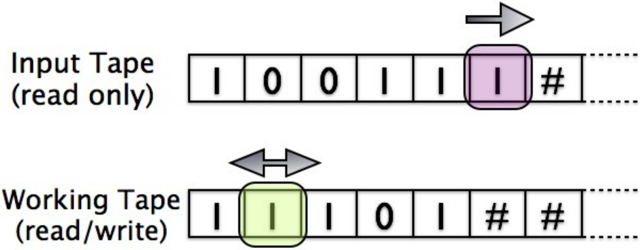
Streaming Model

Deterministic Turing Transducer



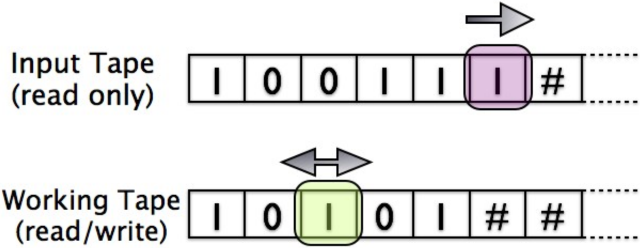
Streaming Model

Deterministic Turing Transducer



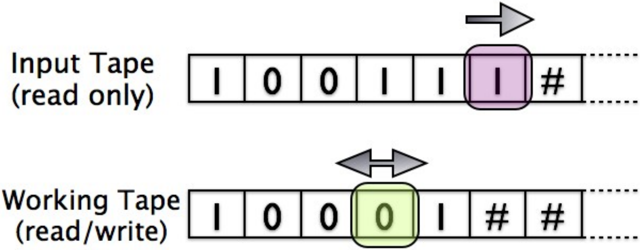
Streaming Model

Deterministic Turing Transducer



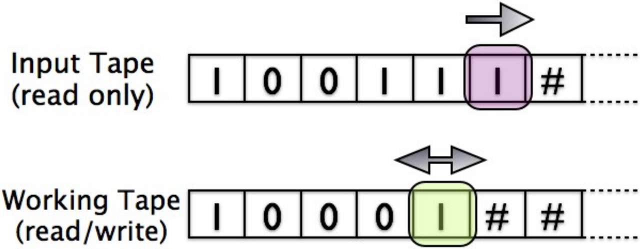
Streaming Model

Deterministic Turing Transducer



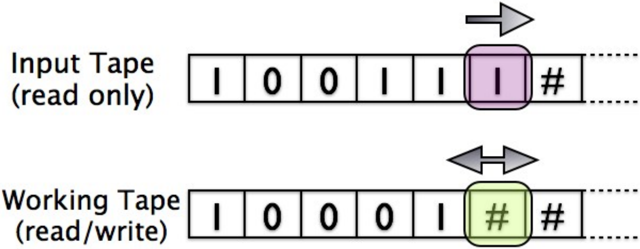
Streaming Model

Deterministic Turing Transducer



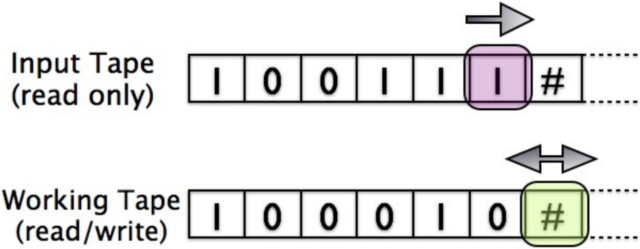
Streaming Model

Deterministic Turing Transducer



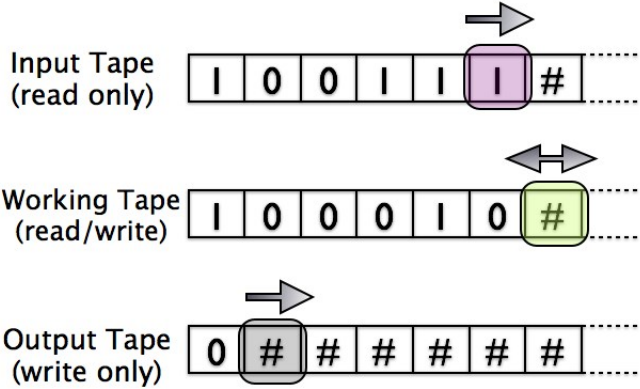
Streaming Model

Deterministic Turing Transducer



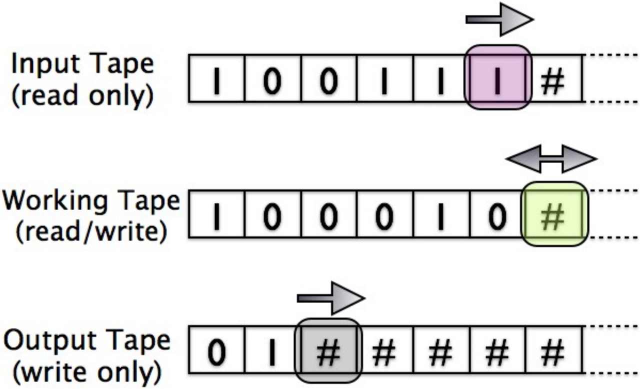
Streaming Model

Deterministic Turing Transducer



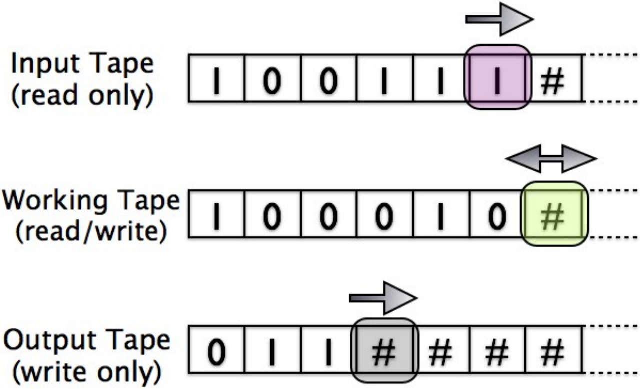
Streaming Model

Deterministic Turing Transducer



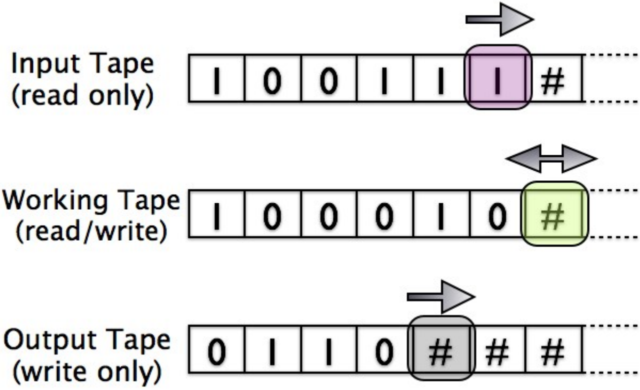
Streaming Model

Deterministic Turing Transducer



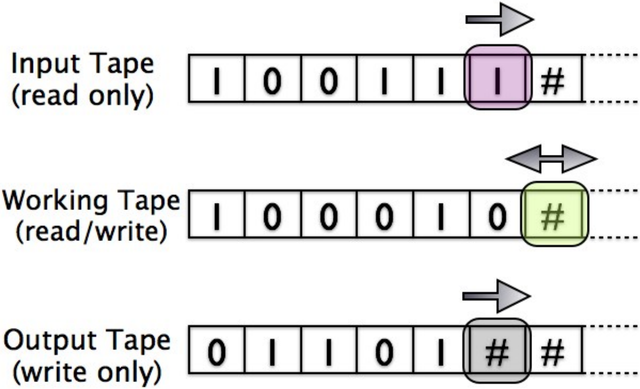
Streaming Model

Deterministic Turing Transducer



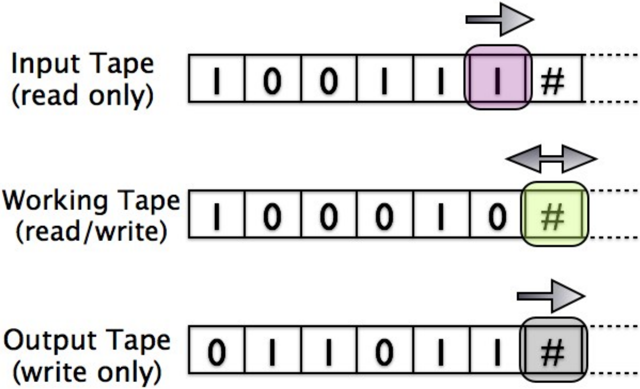
Streaming Model

Deterministic Turing Transducer



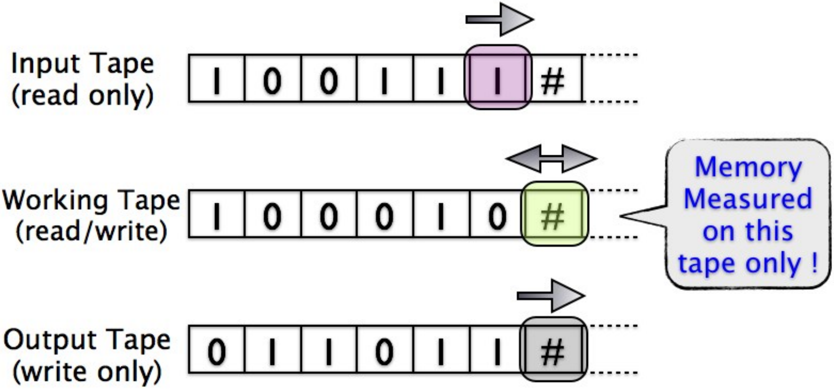
Streaming Model

Deterministic Turing Transducer



Streaming Model

Deterministic Turing Transducer



Bounded Memory Problem – Examples

$T_1 : \begin{cases} a^n b \mapsto b^{n+1} \\ a^n c \mapsto c^{n+1} \end{cases}$	Not bounded memory
$T_2 : \textvisiblespace\textvisiblespace a \textvisiblespace\textvisiblespace\textvisiblespace b \textvisiblespace\textvisiblespace \mapsto \textvisiblespace a \textvisiblespace b$	Bounded memory

Bounded Memory Problem – Examples

$T_1 : \begin{cases} a^n b \mapsto b^{n+1} \\ a^n c \mapsto c^{n+1} \end{cases}$	Not bounded memory
$T_2 : \sqcup\sqcup a \sqcup\sqcup\sqcup b \sqcup\sqcup \mapsto \sqcup a \sqcup b$	Bounded memory

Theorem

For all functional NFT T, the following are equivalent:

1. *T is bounded memory*
2. *T is determinizable*
3. *T satisfies the twinning property.*

Proof based on the following two observations:

1. any DFT is bounded memory
2. bounded memory Turing Transducer $\equiv$ DFT

Closure Properties of Finite State Transducers

Domain, co-domain

The domains and co-domains of NFT are regular.

	T^{-1}	$\overline{T}$	$T_1 \cup T_2$	$T_1 \cap T_2$	$T_1 \circ T_2$
NFT	no	no	yes	no	yes
DFT	no	no	no	no	yes

Table: Closure Properties for NFT and DFT.

Closure Properties of Finite State Transducers

Domain, co-domain

The domains and co-domains of NFT are regular.

	T^{-1}	$\overline{T}$	$T_1 \cup T_2$	$T_1 \cap T_2$	$T_1 \circ T_2$
NFT	no	no	yes	no	yes
DFT	no	no	no	no	yes

Table: Closure Properties for NFT and DFT.

Non-closure by intersection

1. $R(T_1) = \{(a^m b^n, c^m) \mid m, n \geq 0\}$
2. $R(T_2) = \{(a^m b^n, c^n) \mid m, n \geq 0\}$
3. $R(T_1) \cap R(T_2) = \{(a^n b^n, c^n) \mid n \geq 0\}$

Decision problems

Membership $(u, v) \in R(T)$?

Emptiness $R(T) = \emptyset$?

Type checking $T(L_{in}) \subseteq L_{out}$?

Equivalence $R(T_1) = R(T_2)$?

Inclusion $R(T_1) \subseteq R(T_2)$?

	emptiness / membership	type checking (vs NFA)	equiv / inclusion
NFT	PTIME	PSPACE-C	undec
DFT	PTIME	PSPACE-C	PTIME

Table: Decision problems for NFT and DFT.

Undecidability of equivalence and inclusion proved in [Griffiths68].

Functional Finite State Transducers

A transduction (transducer) is functional if each word has at most 1 image.

Theorem (Gurari and Ibarra 83)

Functionality is decidable in PTIME *for* NFT.

Theorem

The equivalence and inclusion of functional NFT is PSPACE-c.

Proof.

T_1 is included in T_2 if and only if

- $dom(T_1) \subseteq dom(T_2)$, and
- $T_1 \cup T_2$ is functional.

$\square$

k-valued Finite State Transducers

A transduction (transducer) is k-valued if each word has at most k images.

Theorem (GI83, Web89, SdS08)

Let $k \in \mathbb{N}$ be fixed.
k-valuedness is decidable in $\mathrm{PT{\scriptsize IME}}$ for NFT.

Theorem (IK86, Web88)

The equivalence and inclusion of k-valued NFT are $\mathrm{PS{\scriptsize PACE}}$-$\mathrm{{\scriptsize C}}$.

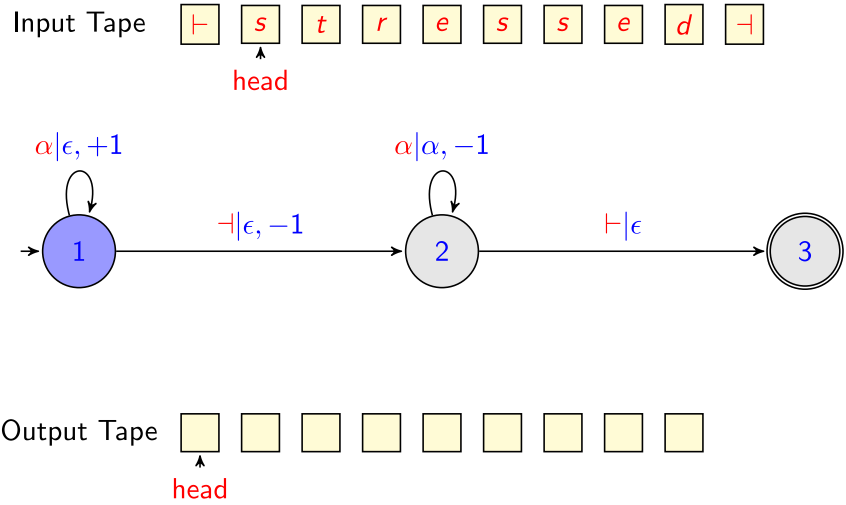
Extensions of NFT

Extensions of NFT

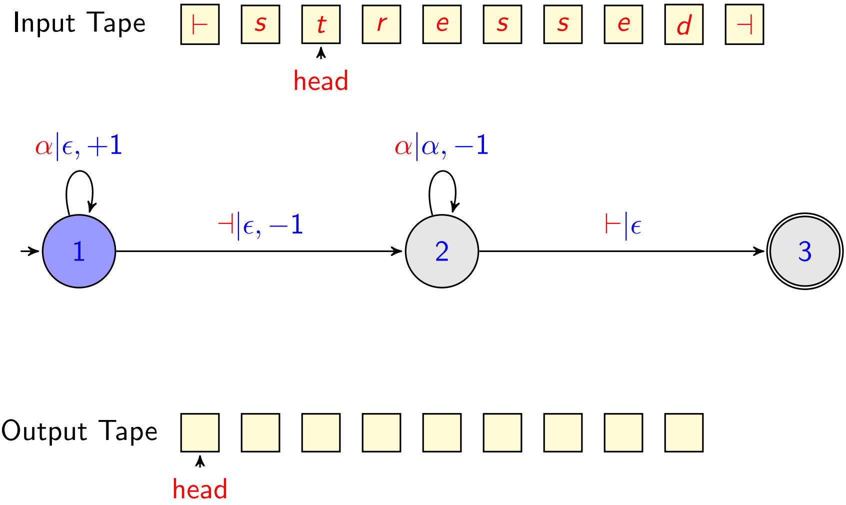
Various more expressive extensions have been considered:

1. two-way input tape
2. string variables (Alur Cerny 2010)
3. pushdown stack

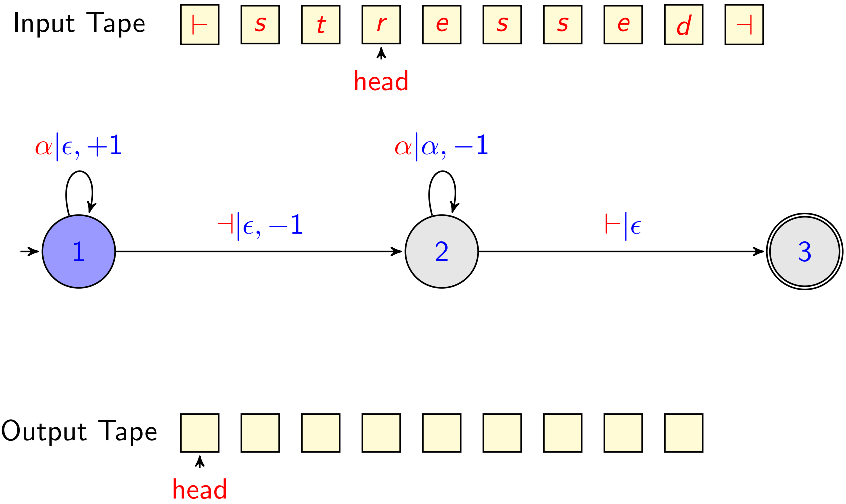
Two-way finite state transducers (2NFT)



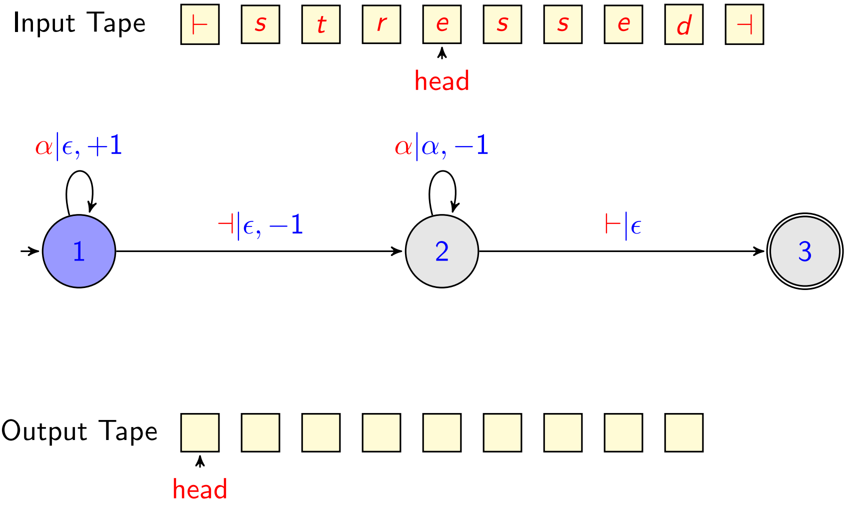
Two-way finite state transducers (2NFT)



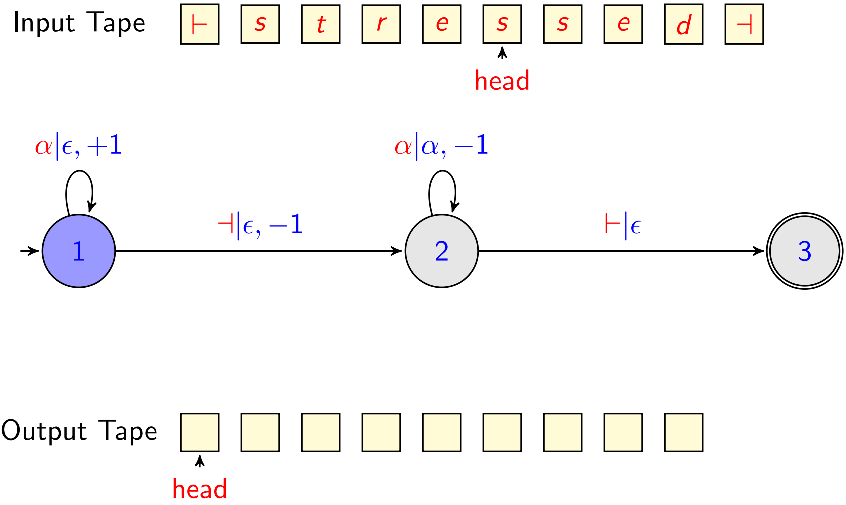
Two-way finite state transducers (2NFT)



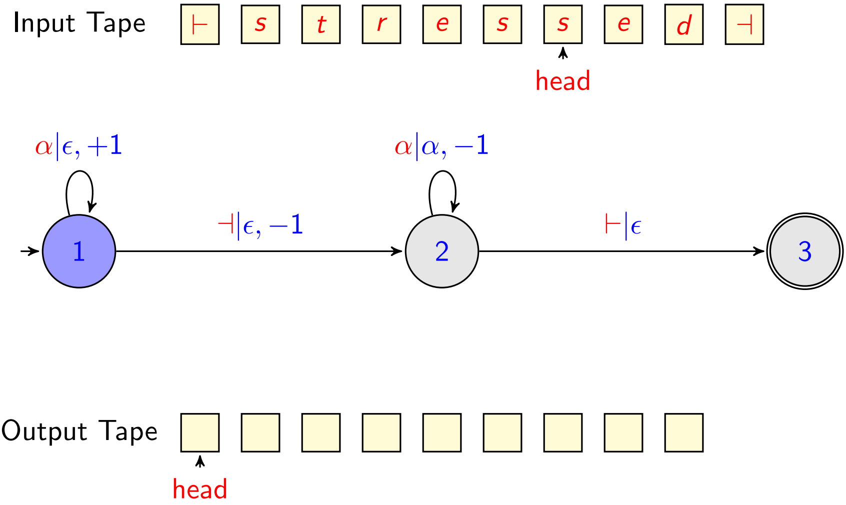
Two-way finite state transducers (2NFT)



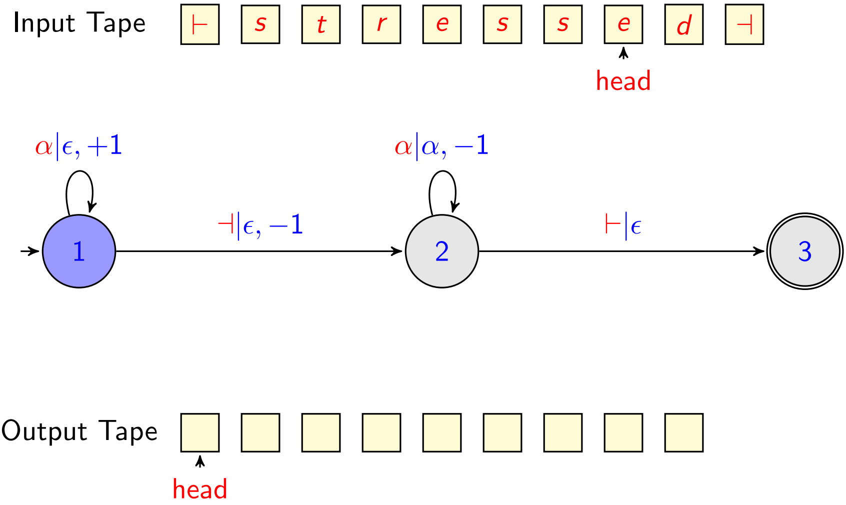
Two-way finite state transducers (2NFT)



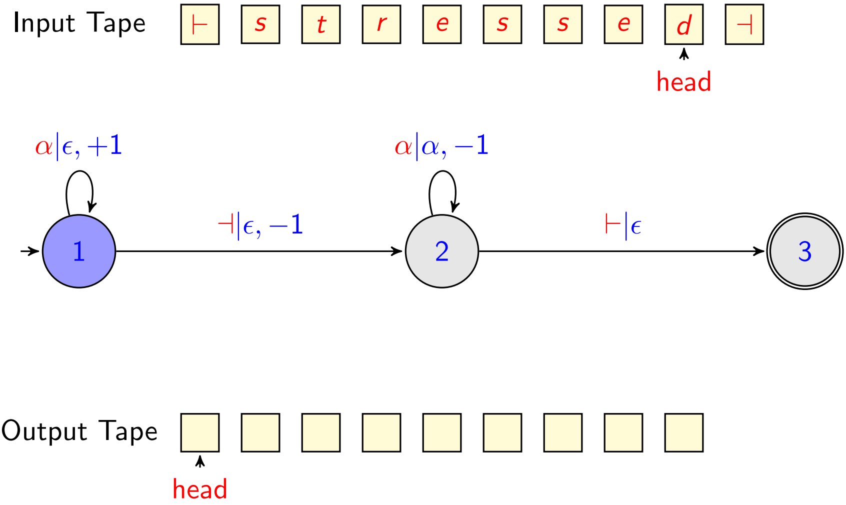
Two-way finite state transducers (2NFT)



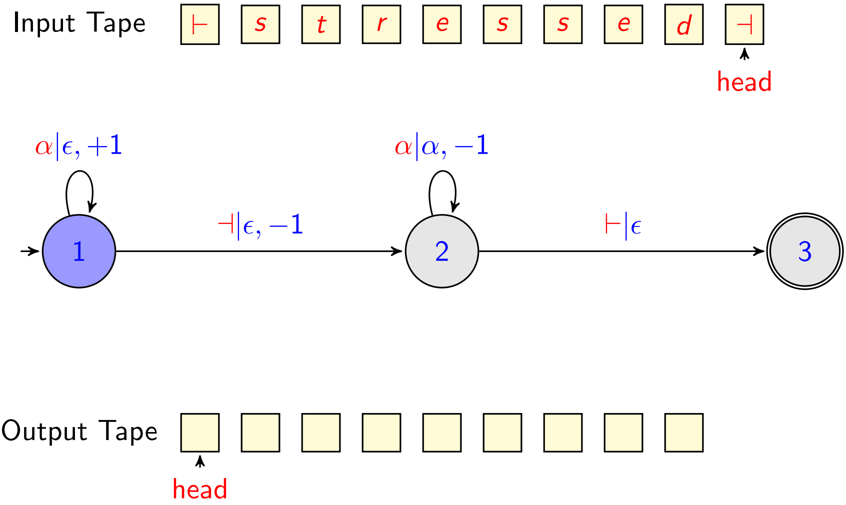
Two-way finite state transducers (2NFT)



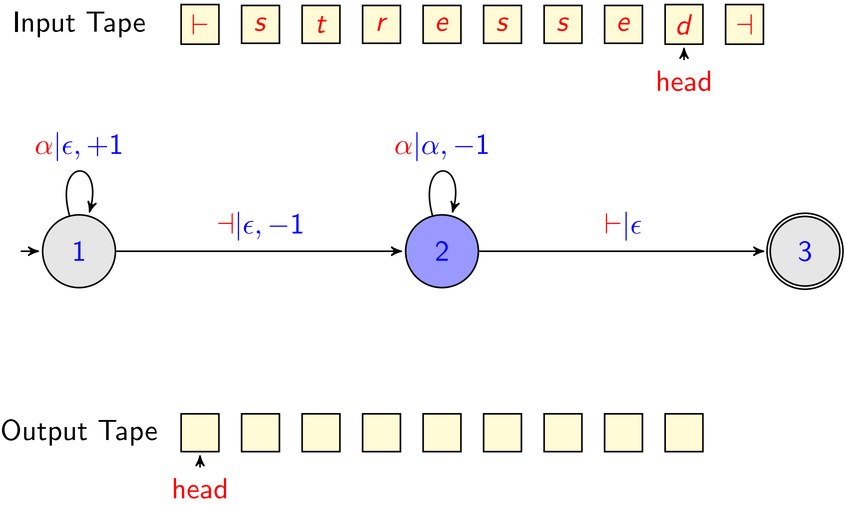
Two-way finite state transducers (2NFT)



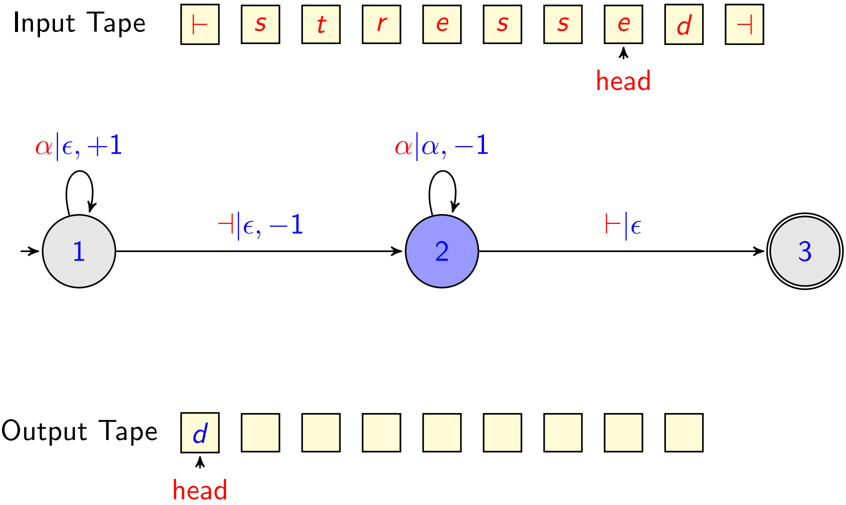
Two-way finite state transducers (2NFT)



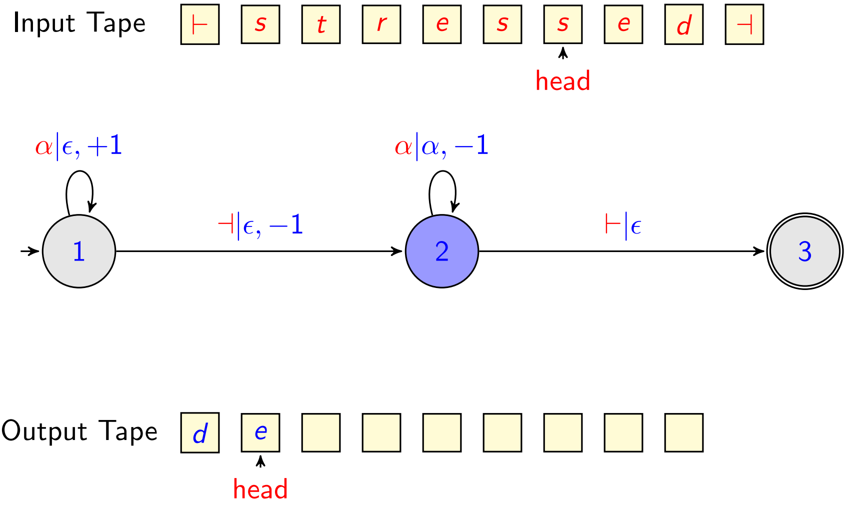
Two-way finite state transducers (2NFT)



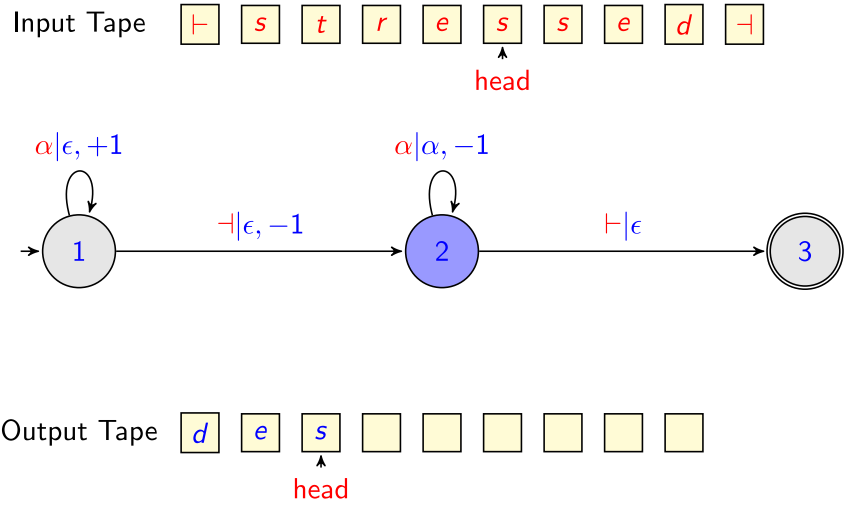
Two-way finite state transducers (2NFT)



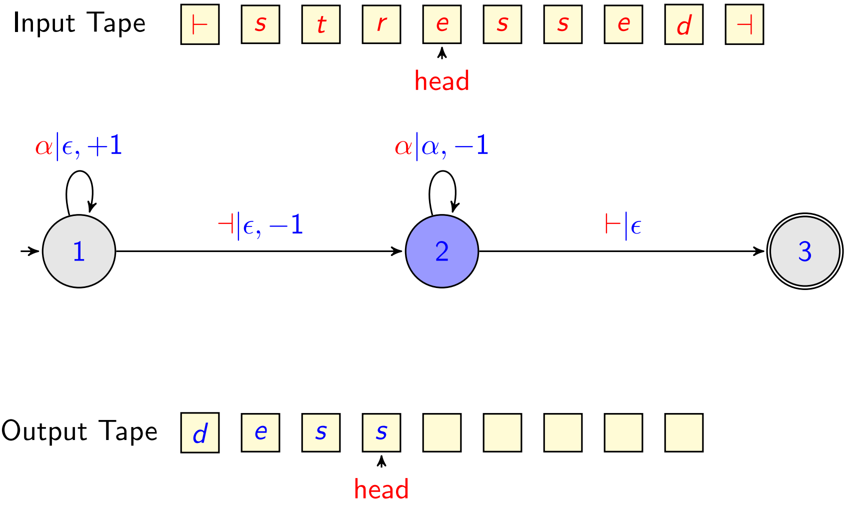
Two-way finite state transducers (2NFT)



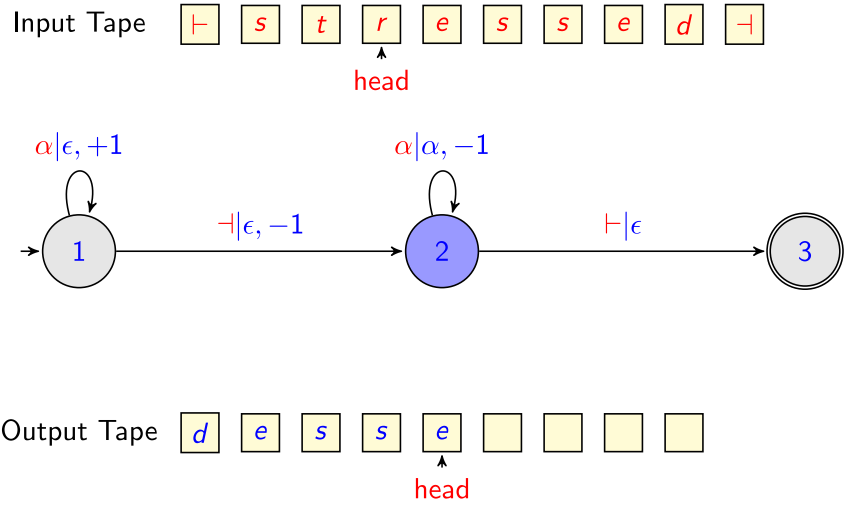
Two-way finite state transducers (2NFT)



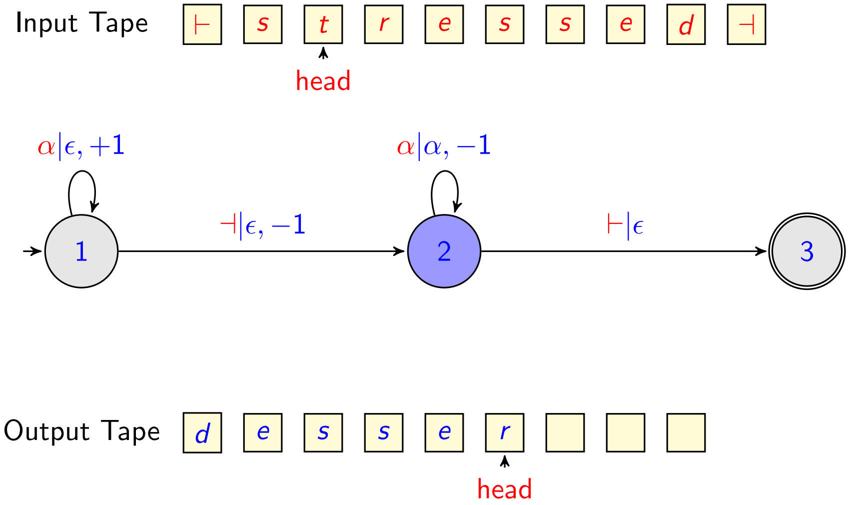
Two-way finite state transducers (2NFT)



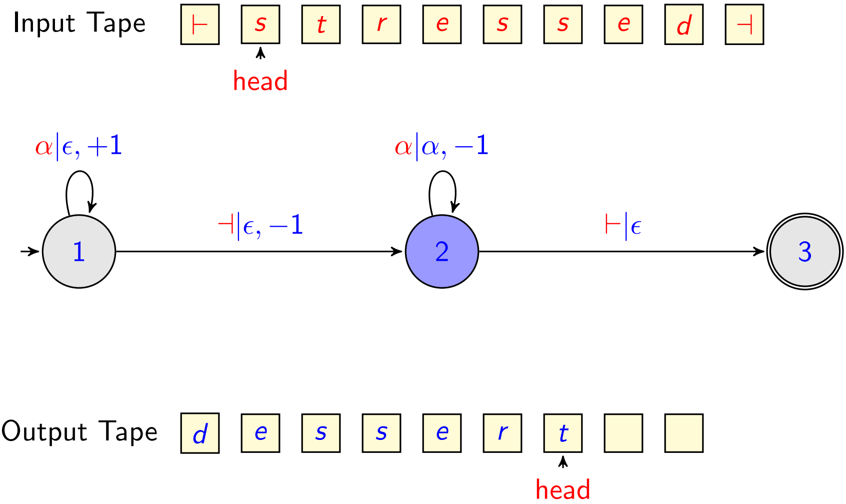
Two-way finite state transducers (2NFT)



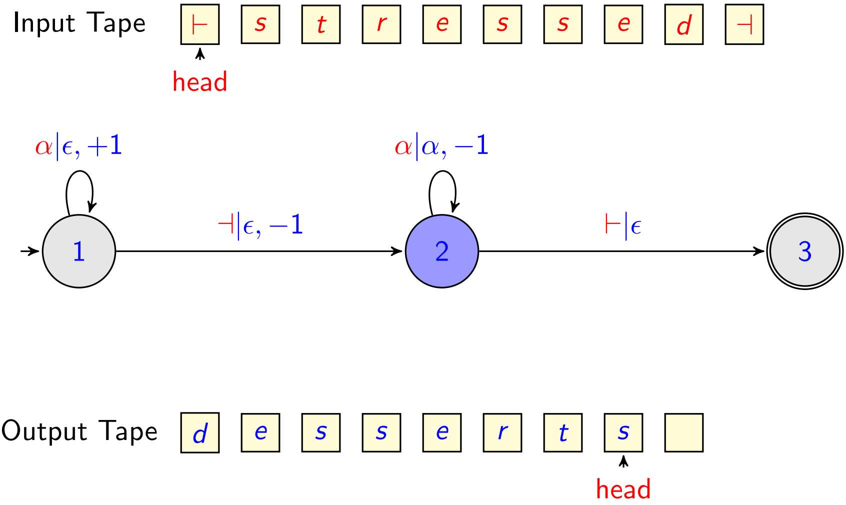
Two-way finite state transducers (2NFT)



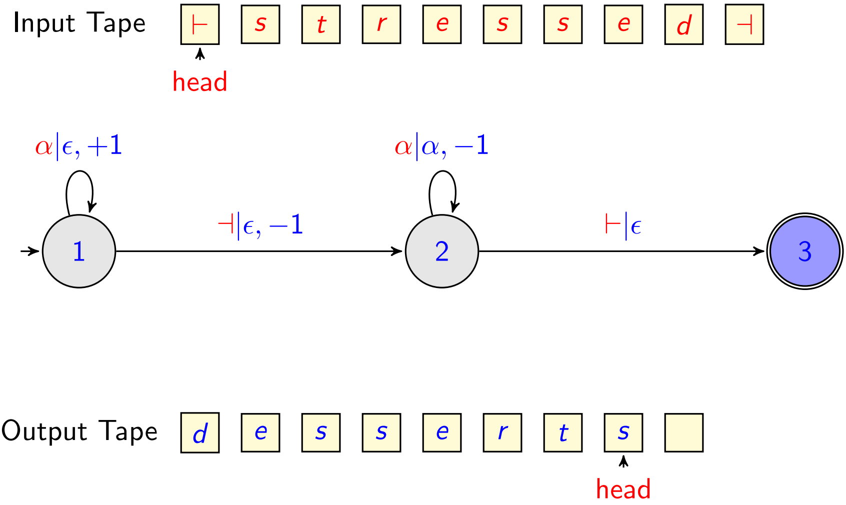
Two-way finite state transducers (2NFT)



Two-way finite state transducers (2NFT)



Two-way finite state transducers (2NFT)



Two-way finite state transducers – Properties

Main Properties of 2NFT

1. still closed under composition (Chytil Jakl 77)
2. equivalence of functional 2NFT is decidable (Culik, Karhumaki, 87)
3. functional 2NFT $\equiv$ 2DFT (Hoogeboom Engelfriet 01, De Souza 13)

Logical Characterization (Hoogeboom Engelfriet 01)

$$\text{2DFT} \equiv \text{MSO transductions}$$

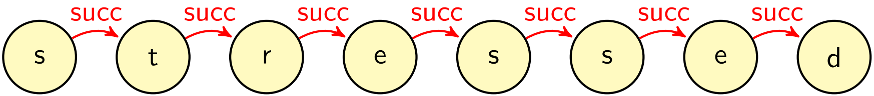
2DFT define regular functions.

MSO Transductions (Courcelle)

- input string seen as the logical structure over $\{succ, (lab_a)_{a\in\Sigma}\}$
- output predicates defined with MSO formulas interpreted over the input structure

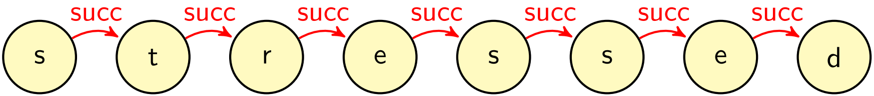
MSO Transductions (Courcelle)

- input string seen as the logical structure over $\{succ, (lab_a)_{a \in \Sigma}\}$
- output predicates defined with MSO formulas interpreted over the input structure



MSO Transductions (Courcelle)

- input string seen as the logical structure over $\{succ, (lab_a)_{a\in\Sigma}\}$
- output predicates defined with MSO formulas interpreted over the input structure

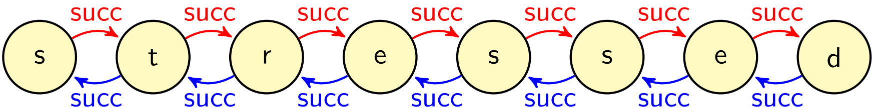


$$\phi_{succ}(x,y) \equiv succ(y,x)$$

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

MSO Transductions (Courcelle)

- input string seen as the logical structure over $\{succ, (lab_a)_{a\in\Sigma}\}$
- output predicates defined with MSO formulas interpreted over the input structure

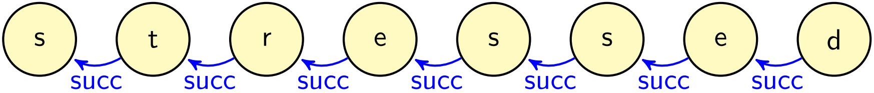


$$\phi_{succ}(x,y) \equiv succ(y,x)$$

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

MSO Transductions (Courcelle)

- input string seen as the logical structure over $\{succ, (lab_a)_{a\in\Sigma}\}$
- output predicates defined with MSO formulas interpreted over the input structure



$$\phi_{succ}(x,y) \equiv succ(y,x)$$

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

Streaming String Transducers (Alur, Cerny, 2010)

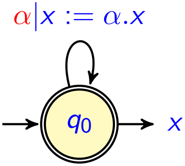
On every transitions, a finite set of variables can be updated by

- appending a string: $x := x.u$
- prepending a string: $x := u.x$
- concatenating two variables: $x := yz$

Streaming String Transducers (Alur, Cerny, 2010)

On every transitions, a finite set of variables can be updated by

- appending a string: $x := x.u$
- prepending a string: $x := u.x$
- concatenating two variables: $x := yz$

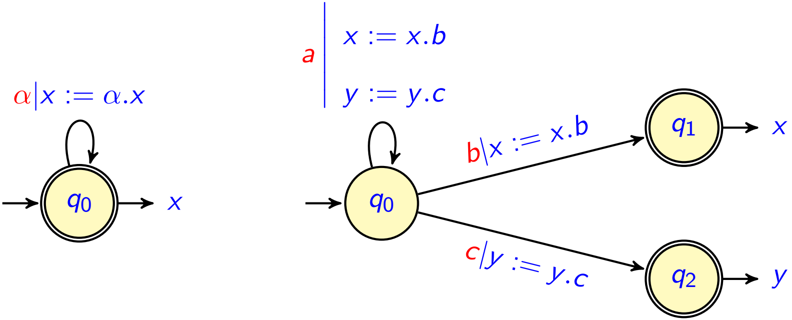


$R(T) = mirror$

Streaming String Transducers (Alur, Cerny, 2010)

On every transitions, a finite set of variables can be updated by

- appending a string: $x := x.u$
- prepending a string: $x := u.x$
- concatenating two variables: $x := yz$



$R(T) = mirror$

$R(T) = a^n\alpha \mapsto \alpha^{n+1}$

Streaming String Transducers

Theorem (Alur Cerny 2010)

The following models are expressively equivalent:

1. *two-way DFT*
2. *MSO transductions*
3. *deterministic (one-way) streaming string transducers with copyless update*

Moreover, SSTs have good algorithmic properties and have been used to analyse list processing programs (Alur Cerny 2011).

Pushdown Transducers

Definition

A pushdown transducer is a pair (A, O) where A is a pushdown automaton and O is an output morphism.

(Bad) Properties

- closure under composition is lost
- Functionality, determinizability, equivalence and inclusion of functional transducers are lost.

Finite State Transducers – Summary

D="(input) deterministic"

f="functional"

DFTs	fNFTs	NFTs
2DFTs	f2NFTs	2NFTs

Finite State Transducers – Summary

D="(input) deterministic"

f="functional"

$\vdash u \dashv \mapsto \text{mirror}(u)$

DFTs fNFTs NFTs

$\cap$ $\cap$ $\cap$

2DFTs f2NFTs 2NFTs

Finite State Transducers – Summary

D="(input) deterministic"

f="functional"

$\{(a,a),(a,b)\}$

DFTs	fNFTs	$\subset$	NFTs
$\cap$	$\cap$		$\cap$
2DFTs	f2NFTs	$\subset$	2NFTs

Finite State Transducers – Summary

D="(input) deterministic"

f="functional"

$a^n\alpha \mapsto \alpha^{n+1}$

DFTs	$\subset$	fNFTs	$\subset$	NFTs
$\cap$		$\cap$		$\cap$
2DFTs		f2NFTs	$\subset$	2NFTs

Finite State Transducers – Summary

D="(input) deterministic"

f="functional"

$$\begin{array}{ccccc} \text{DFTs} & \subsetneq & \text{fNFTs} & \subsetneq & \text{NFTs} \\ \cap & & \cap & & \cap \\ \text{2DFTs} & \equiv & \text{f2NFTs} & \subsetneq & \text{2NFTs} \end{array}$$

[De Souza (13)]

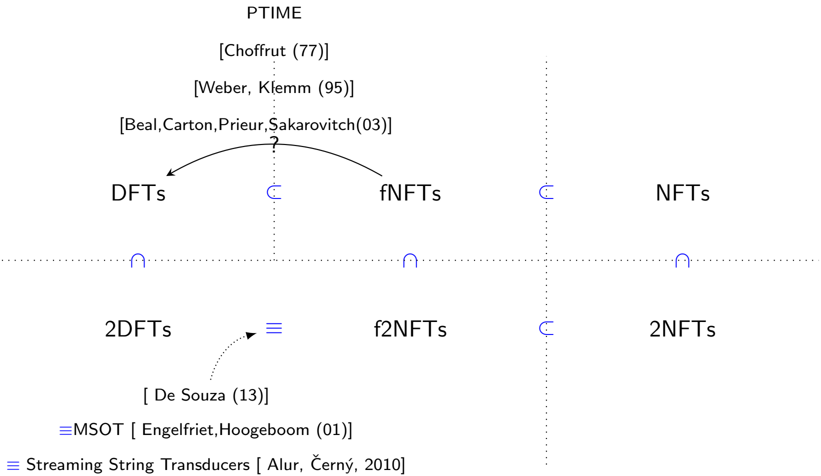
≡MSOT [Engelfriet,Hoogeboom (01)]

≡ Streaming String Transducers [Alur, Černý, 2010]

Finite State Transducers – Summary

D="(input) deterministic"

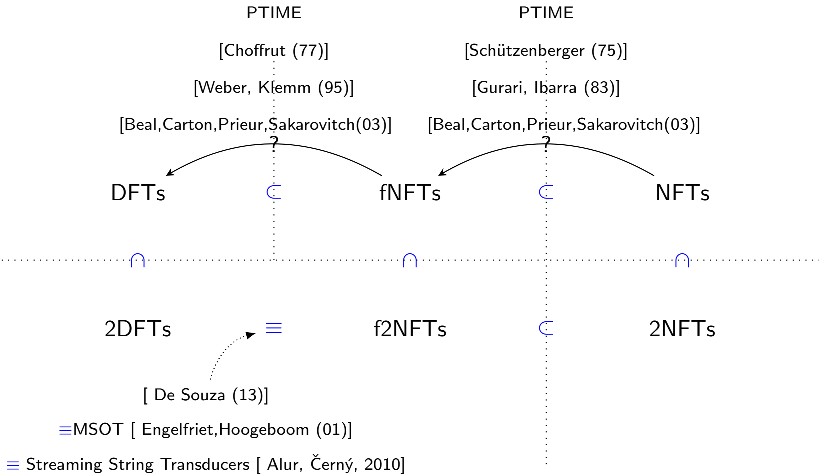
f="functional"



Finite State Transducers – Summary

D="(input) deterministic"

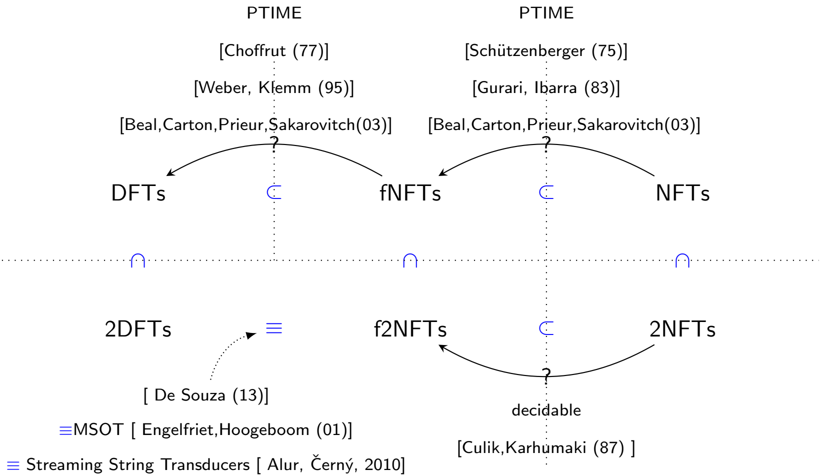
f="functional"



Finite State Transducers – Summary

D="(input) deterministic"

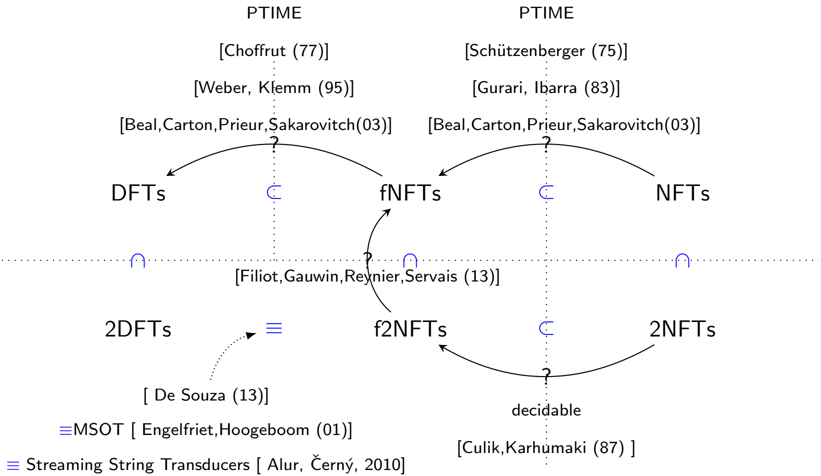
f="functional"



Finite State Transducers – Summary

D="(input) deterministic"

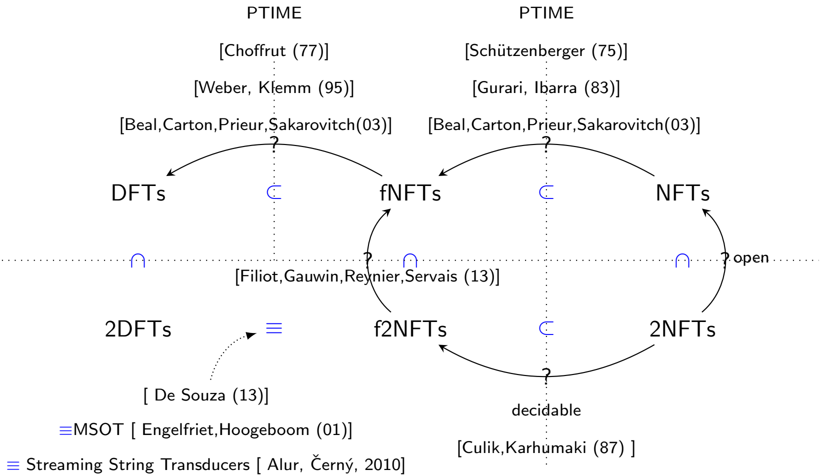
f="functional"



Finite State Transducers – Summary

D="(input) deterministic"

f="functional"



A word about infinite strings

- most transducer models can be extended to (right-) infinite strings
- Büchi / Muller accepting conditions
- most of the results seen so far **still hold with some complications ...**

- determinization of one-way transducers: TP is too strong

- deterministic 2way $<$ functional 2way:

$$T \; : \; u \mapsto \begin{cases} a^\omega \text{ if infinite number of 'a'} \\ u \text{ otherwise} \end{cases}$$

- functional 2way $\equiv$ determinitic 2way $+$ ω-regular look-ahead $\equiv$ ω-MSO transductions $\equiv$ ω-SST (Alur,Filiot,Trivedi,12)

Transducers for Nested Words ($\sim$ Trees)

Motivations

Streaming XML Transformations

- XML are words with a nesting structure
- XML documents can be (very) wide but usually not deep
- in a streaming setting, not reasonable to keep the entire document in memory
- bounded memory streaming transformations ?

Motivations

Streaming XML Transformations

- XML are words with a nesting structure
- XML documents can be (very) wide but usually not deep
- in a streaming setting, not reasonable to keep the entire document in memory
- bounded memory streaming transformations ?

Visibly Pushdown Transducers (VPTs)

- extend Visibly Pushdown Automata (Alur Madhusudan 04)
- well-suited for streaming nested words transformations
- bounded memory analysis for VPT transductions.

Structured Alphabet

Definition (Structured Alphabet)

A structured alphabet, Σ, is a set $\Sigma = \Sigma_c \uplus \Sigma_i \uplus \Sigma_r$, where

- Σ_c are call symbols,
- Σ_i are internal symbols,
- Σ_r, are return symbols.

- a nested word is a word over a structured alphabet

$$c_1 \; c_2 \; a \; r_1$$

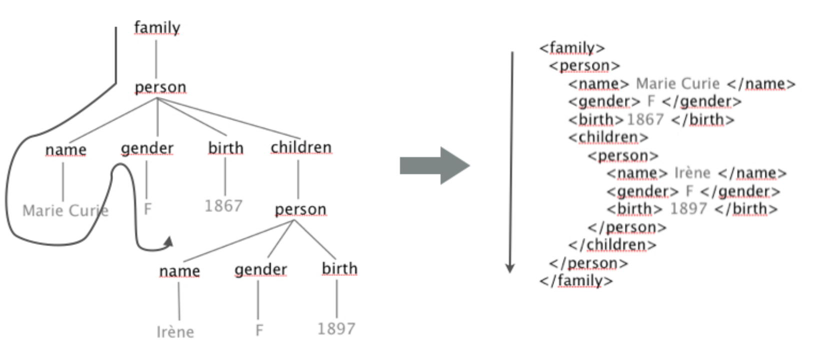
- it is well-nested if there is no pending call nor return symbols

$$c_1 \; c_2 \; a \; r_2 \; b \; r_1$$

Nested Words vs Trees

Encoding

Well-nested words $\equiv$ linearizations of trees



- nested words are well-suited to model tree streams

Visibly Pushdown Automata (VPAs) [Alur,Madhusudan,04]

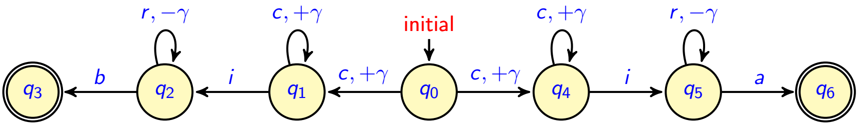
VPAs = Pushdown Automata on structured alphabet
$\Sigma = \Sigma_c \uplus \Sigma_r \uplus \Sigma_i$:

- push **one** stack symbol on call symbols Σ_c
- pop **one** stack symbol on return symbols Σ_r
- don't touch the stack on internal symbols Σ_i
- in this talk, accept on empty stack and final state

Visibly Pushdown Automata (VPAs) [Alur,Madhusudan,04]

VPAs = Pushdown Automata on structured alphabet
$\Sigma = \Sigma_c \uplus \Sigma_r \uplus \Sigma_i$:

- push **one** stack symbol on call symbols Σ_c
- pop **one** stack symbol on return symbols Σ_r
- don't touch the stack on internal symbols Σ_i
- in this talk, accept on empty stack and final state



$$L(A) = \{c^n\ i\ r^n\ a \mid n > 0\} \cup \{c^n\ i\ r^n\ b \mid n > 0\}$$

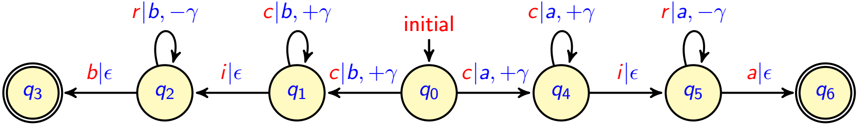
Properties of VPA

- NFA $<$ VPA $<$ PA
- close under all Boolean operations
- NFA algorithmic properties are preserved (equivalence, universality, ...)
- applicatons in
  - computer-aided verification
  - XML processing
- see `http://www.cs.uiuc.edu/~madhu/vpa/`

Visibly Pushdown Transducers (VPTs)

Definition

Pair (A, O) where $A : VPA$ and O is an output morphism.



$$R(T) = \{(c^n\ i\ r^n\ a, a^{2n}) \mid n > 0\} \cup \{(c^n\ i\ r^n\ b, b^{2n}) \mid n > 0\}$$

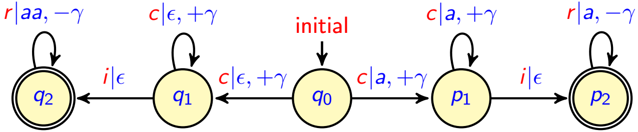
Properties of Visibly Pushdown Transducers

- NFT $<$ VPT $<$ PT
- dVPTs $<$ (functional) VPT
- closed under composition if the output is well-nested
- closed under VPA-lookahead
- functionality is decidable in PTime
- k-valuedness is decidable
- equivalence of functional VPTs is decidable (in PTime of dVPTs)
- decidable typechecking problem (if the output is well-nested)

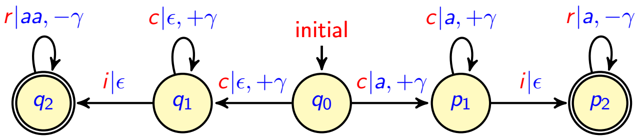
Properties of Visibly Pushdown Transducers

- NFT $<$ VPT $<$ PT
- dVPTs $<$ (functional) VPT
- closed under composition if the output is well-nested
- closed under VPA-lookahead
- functionality is decidable in PTime
- k-valuedness is decidable
- equivalence of functional VPTs is decidable (in PTime of dVPTs)
- decidable typechecking problem (if the output is well-nested)
- **Open Problems**: equivalence of k-valued VPTs, determinizability
- more details in F. Servais's Phd thesis

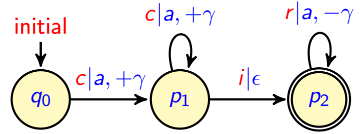
Why is determinizability more difficult?



Why is determinizability more difficult?



It is determinizable by:



but lag increase arbitrarily in (p_1, q_1).

Streamability Problem [F, Gauwin, Reynier, Servais, 11]

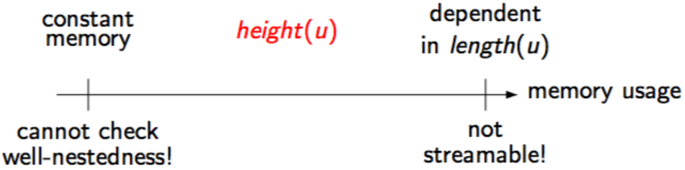
Streaming evaluation: avoid the storage of the whole input

Fix a functional (non-deterministic) VPT T.
How much memory is needed to compute T(u) from an input stream u?

Streamability Problem [F, Gauwin, Reynier, Servais, 11]

Streaming evaluation: avoid the storage of the whole input

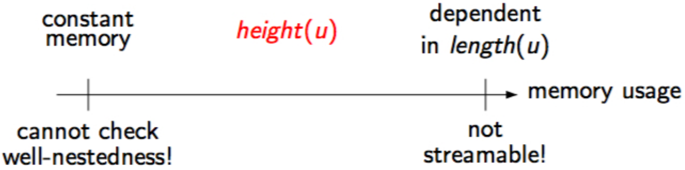
Fix a functional (non-deterministic) VPT T.
How much memory is needed to compute T(u) from an input stream u?



Streamability Problem [F, Gauwin, Reynier, Servais, 11]

Streaming evaluation: avoid the storage of the whole input

Fix a functional (non-deterministic) VPT T.
How much memory is needed to compute T(u) from an input stream u?



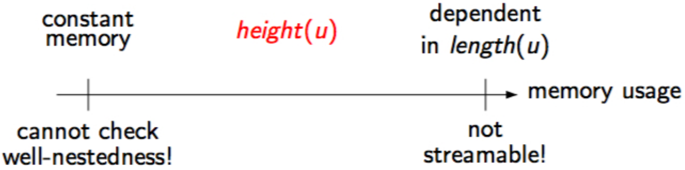
Streamability Problem

Given a VPT T, decide if T defines a transformation that can be evaluated with memory $O(f(height(u))$?

Streamability Problem [F, Gauwin, Reynier, Servais, 11]

Streaming evaluation: avoid the storage of the whole input

Fix a functional (non-deterministic) VPT T.
How much memory is needed to compute T(u) from an input stream u?



Streamability Problem

Given a VPT T, decide if T defines a transformation that can be evaluated with memory $O(f(height(u)))$?

Decidable in NP for VPTs

Determinizability is too strong

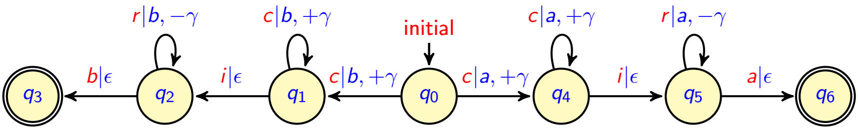
Obs: Deterministic VPTs are always streamable (no output lag)

Determinizability is too strong

Obs: Deterministic VPTs are always streamable (no output lag)

However: determinizable VPTs $<$ streamable VPTs:

$$R(T) : c^n \ i \ r^n \ \alpha \ \mapsto \alpha^{2n} \quad n > 0$$

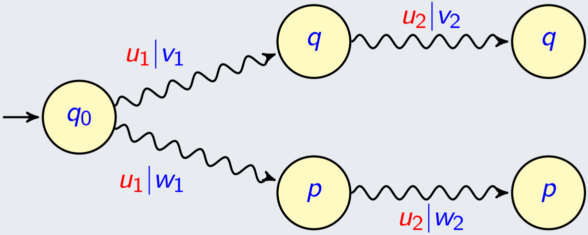


Streamable but not determinizable !

Twinning Property for VPTs

Definition

For all such situations

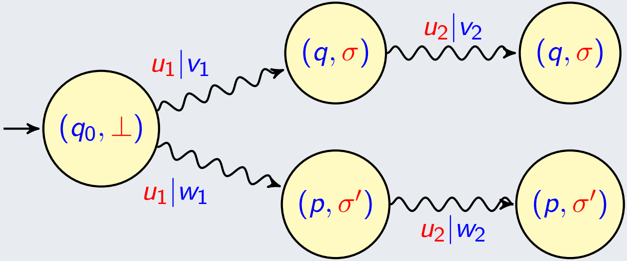


it is the case that $LAG(v_1, w_1) = LAG(v_1 v_2, w_1 w_2)$.

Twinning Property for VPTs

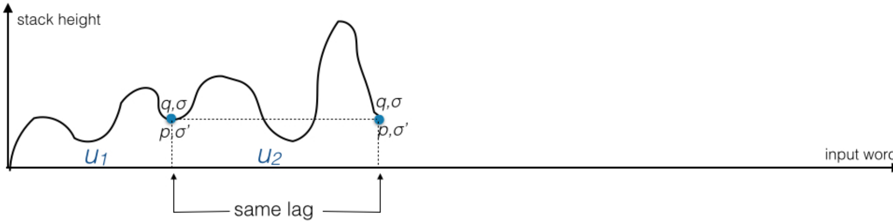
Definition

For all such situations

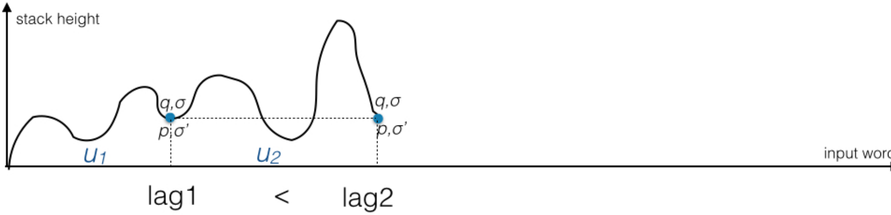


it is the case that $LAG(v_1, w_1) = LAG(v_1 v_2, w_1 w_2)$.

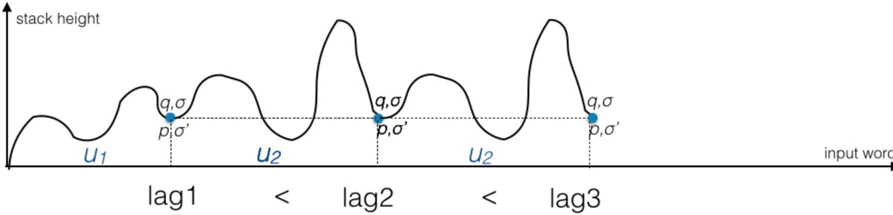
Twinning Property for VPTs



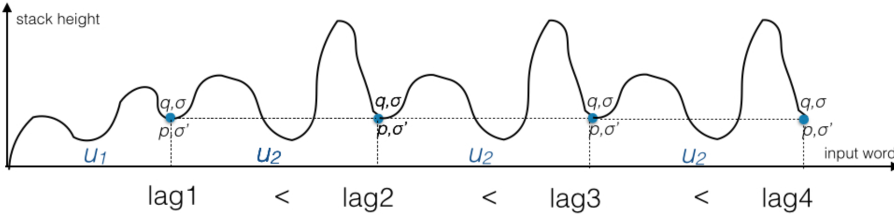
Twinning Property for VPTs



Twinning Property for VPTs



Twinning Property for VPTs



Twinning Property for VPTs

Theorem

Given a functional VPT T, T is streamable iff the twinning property holds.
It can be decided in NPtime.

Twinning Property for VPTs

Theorem

Given a functional VPT T, T is streamable iff the twinning property holds.
It can be decided in NPtime.

- TP is machine-independent: streamable VPTs is class of **transductions**.
- decidability based on reversal-bounded pushdown counter machines
- same result extend to **strongly streamable** (memory depends only on *current height*)

Other tree transducer models

- top-down tree transducers

$$q(f(x_1, \ldots, x_n)) \to C[q_1(x_{i_1}), \ldots, q_p(x_{i_p})]$$

(see TATA[3] book

- macro tree transducers

```
fun q(t1 t2 t3 t4 t)=
if t = a() then
 return F (t1,t2)
else
 if t=g(u,v) then
 return C(q'(t1,t2,u), q''(t3,t4,v))
```

- see Joost Engelfriet and Sebastian Maneth's work

[3] *Tree Automata Techniques and Applications*, tata.gforge.inria.fr

Church Problem

Church Problem (aka Church Synthesis Problem)

Definition (Church 57)

- R a relation, or *requirements*, from a domain D to a domain D'
- synthesize a program P such for all $X \in D$, $(X, P(X)) \in R$.

Church Problem (aka Church Synthesis Problem)

Definition (Church 57)

- R a relation, or *requirements*, from a domain D to a domain D'
- synthesize a program P such for all $X \in D$, $(X, P(X)) \in R$.

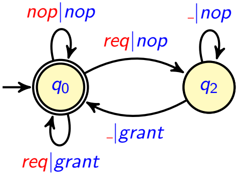
Reactive System Synthesis

Let Σ_{in} and Σ_{out} be to finite alphabets.

- reactive systems continuously react to stimuli produced by some **uncontrollable** environment
- $D = \Sigma_{in}^{\omega}$, $D' = \Sigma_{out}^{\omega}$
- R is a synchronous relation given by a (non-deterministic) symbol-to-symbol Büchi transducer
- P is a Mealy machine (deterministic symbol-to-symbol transducer)

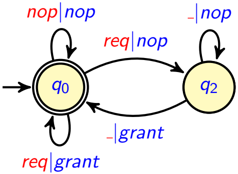
Reactive System Synthesis: Example

- $\Sigma_{in} = \{req, nop\}$
- $\Sigma_{out} = \{grant, nop\}$.
- Requirement R: if there is a request, it must be eventually granted

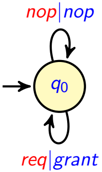


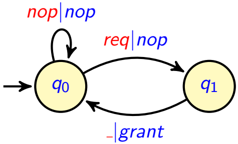
Reactive System Synthesis: Example

- $\Sigma_{in} = \{req, nop\}$
- $\Sigma_{out} = \{grant, nop\}$.
- Requirement R: if there is a request, it must be eventually granted



- Possible programs (Mealy machines) that realize R:





Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in}) :

Player *out* (Σ_{out}) :

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in}) : i_1

Player *out* (Σ_{out}) :

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in}) : i_1

Player *out* (Σ_{out}) : o_1

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in}) : i_1 i_2

Player *out* (Σ_{out}) : o_1

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in})	:	i_1	i_2
Player *out* (Σ_{out})	:	o_1	o_2

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in})	:	i_1	i_2	i_3
Player *out* (Σ_{out})	:	o_1	o_2	

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in})	:	i_1	i_2	i_3
Player *out* (Σ_{out})	:	o_1	o_2	o_3

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in})	:	i_1	i_2	i_3	i_4
Player *out* (Σ_{out})	:	o_1	o_2	o_3	

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in})	:	i_1	i_2	i_3	i_4
Player *out* (Σ_{out})	:	o_1	o_2	o_3	o_4

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in})	:	i_1	i_2	i_3	i_4	i_5
Player *out* (Σ_{out})	:	o_1	o_2	o_3	o_4	

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in})	:	i_1	i_2	i_3	i_4	i_5
Player *out* (Σ_{out})	:	o_1	o_2	o_3	o_4	o_5

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in})	:	i_1	i_2	i_3	i_4	i_5	$\dots$
Player *out* (Σ_{out})	:	o_1	o_2	o_3	o_4	o_5	$\dots$

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

Player in (Σ_{in})	:	i_1	i_2	i_3	i_4	i_5	$\dots$
Player *out* (Σ_{out})	:	o_1	o_2	o_3	o_4	o_5	$\dots$

- **Def:** Player *out* wins if $(i_1 i_2 i_3 \dots, o_1 o_2 o_3 \dots) \in R$.

Church Game

Definition

- **turn-based** game between two players
- Player *in* chooses input symbols in Σ_{in}
- Player *out* chooses output symbols in Σ_{out}
- they play during an infinite number of rounds.

$$\begin{array}{lcllllll} \text{Player in } (\Sigma_{in}) & : & i_1 & i_2 & i_3 & i_4 & i_5 & \dots \\ \text{Player } out \ (\Sigma_{out}) & : & o_1 & o_2 & o_3 & o_4 & o_5 & \dots \end{array}$$

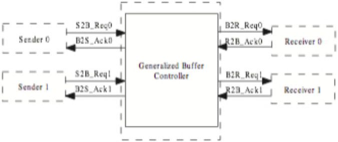
- **Def:** Player *out* wins if $(i_1 i_2 i_3 \dots, o_1 o_2 o_3 \dots) \in R$.
- **Prop:** There exists a program that realizes the requirements R iff Player *out* has a winning strategy.

State of the Art

- reactive system synthesis from ω-regular specifications is decidable (Büchi Landweber 69)
- reactive system synthesis from LTL specifications is 2-ExpTime-c (Pnueli Rosner 89)
- several tools for LTL synthesis:
  - Lily (Jobstmann Bloem 06)
  - Acacia (Filiot Jin Raskin 09)
  - Unbeast (Ehlers 10)
- very active community in game theory for synthesis
  - quantitative games
  - multi-player games
  - stochastic games
  - ...

Acacia Tool

GenBuf spec from IBM




```
################################################
[spec_unit sb_0]
################################################
assume s2b_req0=0;
assume G((s2b_req0=1 * b2s_ack0=0) -> X(s2b_req0=1));
assume G(b2s_ack0=1 -> X(s2b_req0=0));

b2s_ack0=0;
G((s2b_req0=0 * X(s2b_req0=1)) -> X(b2s_ack0=0 * X(F(b2s_ack0=1))));
G((b2s_ack0=0 * X(s2b_req0=0)) -> X(b2s_ack0=0));
G(b2s_ack0=0 + b2s_ack1=0);

################################################
[spec_unit sb_1]
################################################
assume s2b_req1=0;
assume G((s2b_req1=1 * b2s_ack1=0) -> X(s2b_req1=1));
assume G(b2s_ack1=1 -> X(s2b_req1=0));

b2s_ack1=0;
G((s2b_req1=0 * X(s2b_req1=1)) -> X(b2s_ack1=0 * X(F(b2s_ack1=1))));
G((b2s_ack1=0 * X(s2b_req1=0)) -> X(b2s_ack1=0));
G(b2s_ack0=0 + b2s_ack1=0);

################################################
[spec_unit br_0]
################################################
assume r2b_ack0=0;
assume G(b2r_req0=0 -> X(r2b_ack0=0));
assume G(b2r_req0=1 -> X(F(r2b_ack0=1)));

b2r_req0=0;
G(r2b_ack0=1 -> X(b2r_req0=0));
G((b2r_req0=1 * r2b_ack0=0) -> X(b2r_req0=1));
G((b2r_req0=1 * X(b2r_req0=0)) -> X(b2r_req0=0 U (b2r_req0=0 * b2r_req1=
G((b2r_req0=0) + (b2r_req1=0) );
G((s2b_req0=1 + s2b_req1=1) -> X(F(b2r_req0=1 + b2r_req1=1)));

################################################
[spec_unit br_1]
################################################
assume r2b_ack1=0;
assume G(b2r_req1=0 -> X(r2b_ack1=0));
assume G(b2r_req1=1 -> X(F(r2b_ack1=1)));
```

Scalable example

From 1-page long to 4-page long specifications

http://lit2.ulb.ac.be/acaciaplus

How is it related to transducer theory?

- reactive systems are streaming machines
- from a relation R, extract a function f such that:
  1. $dom(R) \subseteq dom(f)$
  2. for all $u \in dom(R)$, $f(u) \in R(u)$.
  3. f is a deterministic symbol-to-symbol transducer
- this problem is known as the uniformization problem in transducer theory
- equivalently, is there a bounded memory (symbol-to-symbol) function f such that $f \subseteq R$ and $dom(R) \subseteq dom(f)$?

Conclusion

Contributions

- finite transducers have good closure and algorithmic properties
- nicely extend to visibly pushdown transducers
- streamability problem $\equiv$ synthesis problem

Open Problems and Future Work

Open problems

- equivalence of k-valued VPTs
- determinizability of VPTs
- extension of streaming results to more expressive transducers, e.g. macro tree transducers
- shift from reactive systems to list processing program synthesis

Publications

- E. Filiot, O. Gauwin, P.-A. Reynier, F. Servais: From Two-Way to One-Way Finite State Transducers, LICS 2013.
- R. Alur, E. Filiot, A. Trivedi: Regular Transformations of Infinite Strings, LICS 2012.
- E. Filiot and F. Servais: Visibly Pushdown Transducers with Look Ahead, SOFSEM 2012.
- E. Filiot, O. Gauwin, P.-A. Reynier and F. Servais: Streamability of Nested Word Transductions, FSTTCS 2011.
- E. Filiot, N. Jin, J.-F. Raskin: Antichain and Compositional Algorithms for LTL Synthesis, FMSD 2011.
- E. Filiot, J.-F. Raskin, P.-A. Reynier, F. Servais and J.-M. Talbot: Properties of Visibly Pushdown Transducers, MFCS 2010.
- J.-F. Raskin, F. Servais: Visibly Pushdown Transducers, ICALP 2008.