From Two-way to One-way Finite State Transducers: a Shepherdson’s Approach

Benjamin Monmege, Pierre-Alain Reynier and Jean-Marc Talbot

LIF, Aix-Marseille

DELTA Kick-off - Feb. 9-10
Outline

1 From two-way to universal transducer
2 Normalizing universal transducer
3 Deciding one-wayness of universal transducers
1 - From two-way to universal transducer
Deterministic Two-way Transducers: Example 1

Reverse:

- $\triangleright \Sigma^* \triangleleft \rightarrow \Sigma^*$
- $\triangleright a_1 \ldots a_n \triangleleft \rightarrow a_n \ldots a_1$

Diagram:

- States: 0, 1, 1', 2, q_F
- Transitions:
  - $0 \rightarrow 1 \rightarrow 1' \leftarrow 2 \leftarrow q_F$
  - $\triangleright \epsilon \rightarrow a | \epsilon$
  - $\epsilon \triangleright b | \epsilon$
  - $\epsilon \triangleright a | a$
  - $\epsilon \triangleright b | b$
  - $\epsilon \triangleright \epsilon$
Deterministic Two-way Transducers: Example 2

LastFirst:
\[
\begin{align*}
\{a, b\}^+ &\xrightarrow{\uparrow} \{a, b, c\}^+  \\
\{a, b\}^+ &\xrightarrow{\uparrow} \{a, b, c\}^+ & \text{with } \sigma \in \{a, b\}
\end{align*}
\]
The One-wayness Problem of D2FST

Definition
A D2FST is *one-way definable* if there exists an equivalent one-way (functional) transducer.

One-way: directions restricted to →

Functional =
  deterministic
  +
  look-ahead (finite information on the suffix of the input)

Note that
- Reverse is not one-way definable
- LastFirst is one-way definable
The One-wayness Problem of D2FST

Definition

A D2FST is *one-way definable* if there exists an equivalent one-way (functional) transducer.

One-way: directions restricted to →

Functional =

- deterministic

  +

  look-ahead (finite information on the suffix of the input)

Note that

- Reverse is not one-way definable
- LastFirst is one-way definable
The One-wayness Problem of D2FST

Definition

A D2FST is *one-way definable* if there exists an equivalent one-way (functional) transducer.

One-way: directions restricted to \(\rightarrow\)

Functional =

- deterministic

+ look-ahead (finite information on the suffix of the input)

Note that

- Reverse is not one-way definable
- LastFirst is one-way definable
Universal Transducers

**Definition**

Universal transducer = word-to-word finite state transducers that are deterministic for some (co-deterministic) look-ahead automaton

aka Deterministic top-down tree-to-string transducers with look-ahead and monadic inputs (see Maneth-Engelfriet)

Transition rules for some universal transducer $G$ with look-ahead automaton $B$:

$q(\sigma) \xrightarrow{p} v_0 q_1 v_1 \ldots q_n v_n$

$\sigma \in \Sigma$, $v_0, \ldots v_n \in \Delta^*$, States $q, q_1 \ldots q_n$ of $G$, a state $p$ from $B$

+ some initialization rules $\tau_I : p \rightarrow v_0 q_1 v_1 \ldots q_n v_n$. 

Universal Transducers

**Definition**

Universal transducer $= \text{word-to-word finite state transducers that are deterministic for some (co-deterministic) look-ahead automaton}

aka Deterministic top-down tree-to-string transducers with look-ahead and monadic inputs (see Maneth-Engelfriet)

Transition rules for some universal transducer $\mathcal{G}$ with look-ahead automaton $B$:

$q(\sigma) \xrightarrow{p} v_0 q_1 v_1 \ldots q_n v_n$

$\sigma \in \Sigma$, $v_0, \ldots v_n \in \Delta^*$, States $q, q_1 \ldots q_n$ of $\mathcal{G}$, a state $p$ from $B$

+ some initialization rules $\tau_I : p \rightarrow v_0 q_1 v_1 \ldots q_n v_n$. 
Universal Transducers

**Definition**

Universal transducer = word-to-word finite state transducers that are deterministic for some (co-deterministic) look-ahead automaton

aka Deterministic top-down tree-to-string transducers with look-ahead and monadic inputs (see Maneth-Engelfriet)

Transition rules for some universal transducer $G$ with look-ahead automaton $B$:

$q(\sigma) \xrightarrow{p} v_0 q_1 v_1 \ldots q_n v_n$

$\sigma \in \Sigma$, $v_0, \ldots v_n \in \Delta^*$, States $q, q_1 \ldots q_n$ of $G$, a state $p$ from $B$ + some initialization rules $\tau_I : p \rightarrow v_0 q_1 v_1 \ldots q_n v_n$.  


Universal Transducers

**Definition**

Universal transducer = word-to-word finite state transducers that are deterministic for some (co-deterministic) look-ahead automaton

aka Deterministic top-down tree-to-string transducers with look-ahead and monadic inputs (see Maneth-Engelfriet)

Transition rules for some universal transducer $G$ with look-ahead automaton $B$:

$$ q(\sigma) \xrightarrow{p} v_0 q_1 v_1 \ldots q_n v_n $$

$\sigma \in \Sigma, v_0, \ldots v_n \in \Delta^*$, States $q, q_1 \ldots q_n$ of $G$, a state $p$ from $B$

+ some initialization rules $\tau_I : p \rightarrow v_0 q_1 v_1 \ldots q_n v_n$. 
Universal Transducers: Semantics by Example

Reverse:
- Look-ahead automaton
- A universal transducer $G$

$$\tau_I(p) = q$$

Semantics: $G(\triangleright aabba\triangleleft) = [q]_p(aabba\triangleleft) = abbaa$

$$(aabba\triangleleft, q, p) \vdash (abba\triangleleft, qa, p) \vdash (bba\triangleleft, qaa, p) \vdash (ba\triangleleft, qbaa, p)$$

$$\vdash (a\triangleleft, qbbaa, p) \vdash (\triangleleft, qabbaa, p) \vdash (\epsilon, abbaa, p_F)$$

At each step, the suffix of the input word belongs to $L(p)$.
Universal Transducers: Semantics by Example

Reverse:

\[
\begin{align*}
\text{Look-ahead automaton} \\
\text{A universal transducer } G
\end{align*}
\]

\[
\tau_I(p) = q
\]

\[
G(\triangleright aabba\triangleleft) = \llbracket q \rrbracket_p(aabba\triangleleft) = abbaa
\]

\[
(aabba\triangleleft, q, p) \vdash (aabba\triangleleft, qa, p) \vdash (baa\triangleleft, qaa, p) \vdash (baa\triangleleft, qbaa, p)
\]

\[
\vdash (a\triangleleft, qbbaa, p) \vdash (a\triangleleft, qabbaa, p) \vdash (\epsilon, abbaa, p_F)
\]

At each step, the suffix of the input word belongs to \( L(p) \)
Universal Transducers: Semantics by Example (II)

LastFirst: \[ \langle \{a, b\} \rangle^+ \rightarrow \langle \{a, b, c\} \rangle^+ \] with \( \sigma \in \{a, b\} \)

\[ a, b \]
\[ p \] \[ \rightarrow a, b \] \[ p' \] \[ \triangleleft \] \[ p_F \]

\[ \tau_I(p) = \langle q \rangle \]

\[ q(a) \xrightarrow{p} q c \quad q(b) \xrightarrow{p} q c \]
\[ q(a) \xrightarrow{p'} q' a \quad q(b) \xrightarrow{p'} q' b \]
\[ q'(\langle \rangle) \rightarrow \epsilon \]

\[ [q]_p = \{ [q]_p(u) | u \in L(p) \} = ac^* + bc^* \quad [q']_{p'} = \epsilon \]
The (co-deterministic) look-ahead construction

Left-left traversals on $u$ in $A = (q, q')$ if $u$ admits a left-left computation in the D2FST $A$ from $(q, \rightarrow)$ to $(q', \leftarrow)$

$\mathbb{T}$ : the set of the finite set of all left-left traversals

- The set of states of the look-ahead automaton is $\mathbb{T}$
- Transitions are defined from $\mathbb{T} \times \Sigma \times \mathbb{T}$
From D2FST to Universal Transducers

The (co-deterministic) look-ahead construction

Traversals $T = \{(q_1, q_2), (q_3, q_4)\}$

Transitions for $a = \begin{cases} (q, \to, a, q_1, \to, v_0) & (q_2, \leftarrow, a, q', \to, v_1) \\ (q', \to, a, q_3, \to, v_2) & (q_4, \leftarrow, a, q'', \to, v_3) \\ (r, \to, a, r', \leftarrow, v_4) & (r', \leftarrow, a, r'', \leftarrow v_5) \end{cases}$

New traversal $T' = \{(q, q''), (r, r'')\}$ and a transition $(T', a, T)$ in the look-ahead.
The (co-deterministic) look-ahead construction

Traversals \( T = \{(q_1, q_2), (q_3, q_4)\} \)

Transitions for \( a = \)

New traversal \( T' = \{(q, q''), (r, r'')\} \) and a transition \((T', a, T)\) in the look-ahead.
The (co-deterministic) look-ahead construction

Traversals $T = \{(q_1, q_2), (q_3, q_4)\}$

Transitions for $a = \{\}

New traversal $T' = \{(q, q''), (r, r'')\}$ and a transition $(T', a, T)$ in the look-ahead.
The transition rules construction

We produce a transition rule for each \( a \in \Sigma \), for each pair \((q, q'')\) and each look-ahead \( T \) such that \((q, q'') \in T'\) and \((T', a, T)\) is a look-ahead rule. E.g.

\[ T = \{(q_1, q_2), (q_3, q_4)\} \]
The transition rules construction

We produce a transition rule for each \( a \in \Sigma \), for each pair \((q, q'')\) and each look-ahead \( T \) such that \((q, q'') \in T'\) and \((T', a, T)\) is a look-ahead rule. E.g.

\[ T = \{(q_1, q_2), (q_3, q_4)\} \]

\[
\begin{align*}
(q'', \leftarrow) &\rightarrow v_3 q_4 \\
(q', \rightarrow) &\rightarrow v_2 q_3 \\
(q, \rightarrow) &\rightarrow v_1 q_2 \\
&\rightarrow v_0 (q_1, q_2) v_1 v_2 (q_2, q_3) v_3
\end{align*}
\]
2 - Normalizing universal transducer
A language $[q]_p$ is

**periodic** if for $v \in \Sigma^*$,

$$[q]_p \subseteq \{v\}^*$$

the shortest of such $v$ is the period $\pi$ of $[q]_p$

**almost-periodic** if for some $\beta, \pi \in \Sigma^*$,

$$[q]_p \subseteq \beta.\{\pi\}^*$$

**semi-periodic** if for some $k \in \mathbb{N}$, some $\{(\beta_i, \pi_i) \mid i \in [1..k]\}$,

$$[q]_p \subseteq \bigcup_k \beta_i.\{\pi_i\}^*$$

**strongly aperiodic** (and thus, infinite) if it is not semi-periodic
A universal transducer $\mathcal{G}$ with look-ahead automaton $B$

What?

*Find an equivalent universal transducer $\mathcal{G}'$ s.t. for all $q, p$,*

$$[q]_p \text{ is either infinite periodic or strongly aperiodic}$$

How?

*Essentially modify the look-ahead automaton $B$ (and $\mathcal{G}$ slightly)*
Normalization : Example 1

Reverse

\[ \tau_I(p) = q \]

\[ q(a) \xrightarrow{p} qa \]
\[ q(b) \xrightarrow{p} qb \]
\[ q(\langle) \rightarrow \epsilon \]

\[ \llbracket q \rrbracket_p = \{ a, b \}^* \] is strongly aperiodic

This universal transducer is normalized
Normalization : Example 2

\[
\text{LastFirst} : \quad \triangleright\{\,a, b\,\}^+ \triangleleft \quad \rightarrow \quad \triangleright\{\,a, b, c\,\}^+ \triangleleft \quad \text{with } \sigma \in \{ \,a, b\, \}
\]

\[
\begin{array}{ccc}
\text{p} & \xrightarrow{a, b} & \text{p}' \\
\end{array}
\]

\[
\tau(p) = q
\]

\[
\begin{align*}
\text{q(a)} & \xrightarrow{p} q \ c \\
\text{q(b)} & \xrightarrow{p} q \ c \\
\text{q'(a)} & \xrightarrow{p'_f} q' \ a \\
\text{q'(c)} & \xrightarrow{p'_f} q' \ b \\
\text{q'(\triangleleft)} & \xrightarrow{} \epsilon
\end{align*}
\]

\[
[q]_p = ac^* + bc^* \text{ is semi-periodic}
\]
Removing semi-periodic states:
Look-ahead refinement by splitting states according to the sum of almost-periodic languages

\[ [q]_P = ac^* + bc^* \]

\( p \) is splitted into \( p_a \) and \( p_b \)

\[ \tau_I(p_a) = \triangleright q \triangleleft \]
\[ \tau_I(p_b) = \triangleright q \triangleleft \]

\[ [q]_{p_a} = ac^* \quad [q]_{p_b} = bc^* \]
Removing semi-periodic states:
Look-ahead refinement by splitting states according to the sum of almost-periodic languages

$$\llbracket q \rrbracket_p = ac^* + bc^*$$

$p$ is splitted into $p_a$ and $p_b$

$\tau_I(p_a) = \triangleright q \triangleleft$
$\tau_I(p_b) = \triangleright q \triangleleft$

$$q(a) \xrightarrow{p_a} q \ c$$
$$q(b) \xrightarrow{p_a} q \ c$$
$$q'(\triangleleft) \rightarrow \epsilon$$

$$q(a) \xrightarrow{p_b} q \ c$$
$$q(b) \xrightarrow{p_b} q \ c$$

$$q(a) \xrightarrow{p'} q' \ a$$
$$q(b) \xrightarrow{p'} q' \ b$$

$$\llbracket q \rrbracket_{p_a} = ac^*$$
$$\llbracket q \rrbracket_{p_a} = bc^*$$
Removing almost-periodic states:

- identify the first letter \( \sigma_{q,p} \) of the lcp’s of all \( \llbracket q \rrbracket_p \)
- refine the look-ahead and then modify the transition rules to output \( \sigma_{q,p} \) as soon as possible
- reiterate (lcp of the \( \llbracket q \rrbracket_p \) are shorter)

LastFirst:

\[
\begin{align*}
\tau_I(p_a) &= \triangleright q \triangleleft \\
\tau_I(p_b) &= \triangleright q \triangleleft \\
q(a) \xrightarrow{p_a} q \ c & \quad q(c) \xrightarrow{p_a} q \ c \\
q(a) \xrightarrow{p_b} q \ c & \quad q(c) \xrightarrow{p_b} q \ c \\
q'(\triangle) & \rightarrow \epsilon \\
q'(b) & \rightarrow q' \ b
\end{align*}
\]
Removing almost-periodic states:

\[ \sim \text{earliest output of longest common prefix (lcp)} \]

- Identify the first letter \( \sigma_{q,p} \) of the lcp's of all \( \lbrack q \rbrack_p \)
- Refine the look-ahead and then modify the transition rules to output \( \sigma_{q,p} \) as soon as possible
- Reiterate (lcp of the \( \lbrack q \rbrack_p \) are shorter)

LastFirst: Look-ahead is unchanged

\[
\begin{align*}
\tau_I(p_a) &= \triangleright a \ q \triangleleft \\
\tau_I(p_b) &= \triangleright b \ q \triangleleft \\
q(a) \xrightarrow{p_a} q \ c & \quad q(b) \xrightarrow{p_a} q \ c \\
q(a) \xrightarrow{p_b} q \ c & \quad q(b) \xrightarrow{p_b} q \ c \\
q(a) \xrightarrow{p'} \epsilon & \quad q(b) \xrightarrow{p'} \epsilon
\end{align*}
\]

\( \lbrack q \rbrack_{p_a}, \lbrack q \rbrack_{p_b} \) are periodic of period \( c \)
3 - Deciding one-wayness of universal transducers
Ideas borrowed from [Laurence, Lemay, Niehren, Staworko and Tommasi: "Learning Sequential Tree-to-Word Transducers"]

For a word \( v \) and a language \( L \),

\[
\text{Push}(L, v) = L' \quad \text{if} \quad L.v = v.L'
\]

- \( L \) is periodic of period \( \pi \) then \( v = \pi^k\pi' \) for \( k \in \mathbb{N} \) and \( \pi = \pi'\pi'' \) and \( L' = (\pi')^{-1}(L.\pi') \).

- \( L \) is strongly aperiodic then the set of \( v \)'s is finite and prefix-closed and \( L' = v^{-1}(L.v) \)
Ideas borrowed from [Laurence, Lemay, Niehren, Staworko and Tommasi: "Learning Sequential Tree-to-Word Transducers"]

For a word \( v \) and a language \( L \),

\[
\text{Push}(L, v) = L' \quad \text{if} \quad L.v = v.L'
\]

- \( L \) is periodic of period \( \pi \) then \( v = \pi^k\pi' \) for \( k \in \mathbb{N} \) and \( \pi = \pi'\pi'' \) and \( L' = (\pi')^{-1}(L.\pi') \).
- \( L \) is strongly aperiodic then the set of \( v \)'s is finite and prefix-closed and \( L' = v^{-1}(L.v) \)

\[
\text{Push}(\{\epsilon, ab, ababab, ababababab\}, aba) = \{\epsilon, ba, bababa, bababababa\}
\]

\[
\text{Push}((ab)^*, a) = (ba)^*
\]

\[
\text{Push}(a\Sigma^*, a) = \Sigma^*a
\]
Ideas borrowed from [Laurence, Lemay, Niehren, Staworko and Tommasi: "Learning Sequential Tree-to-Word Transducers"]

For a word \( v \) and a language \( L \),

\[
\text{Push}(L, v) = L' \quad \text{if} \quad L \cdot v = v \cdot L'
\]

- \( L \) is periodic of period \( \pi \) then \( v = \pi^k \pi' \) for \( k \in \mathbb{N} \) and \( \pi = \pi' \pi'' \) and \( L' = (\pi')^{-1}(L \cdot \pi') \).
- \( L \) is strongly aperiodic then the set of \( v \)'s is finite and prefix-closed and \( L' = v^{-1}(L \cdot v) \)

### Pushing Words through States under Look-ahead

Annotating states with (part of) words that are pushed through

Define syntactically \( \text{Push}(\llbracket q \rrbracket_p, v) \) by means of a (finitely many) new states \( q_{|\bar{v}} \) together with its transition rules such that

\[
\text{Push}(\llbracket q \rrbracket_p, v) = \llbracket q_{|\bar{v}} \rrbracket_p
\]
The Algorithm in a Nutshell

Compute from a universal transducer $G$ with $m$ copies a one-way deterministic transducer $A$ with look-ahead.

Ingredient:
- The look-ahead of $A$ is the look-ahead of $G$
- States from $A$ are of the form $[q_1 v_1 \ldots q_n v_n]$ with $n \leq m$ and such that
  - the $q_i$’s are states of $G$
  - the $v_i$’s remain small
  - compute new states with the semantics of universal transducers and Pushing

Small ?
Not larger than $F(|G|)$
Running the Algorithm on Examples (I)

Reverse

\[ \tau_I(p) = q \]

\[ q(a) \xrightarrow{p} qa \]
\[ q(b) \xrightarrow{p} qb \]
\[ q(\triangle) \xrightarrow{} \epsilon \]

\((s_I, \rightarrow, △, p, [q], \rightarrow, \epsilon)\)
\(([q], \rightarrow, a, p, [qa], \rightarrow, \epsilon)\)
\(([qa], \rightarrow, a, p, [qaa], \rightarrow, \epsilon)\)
\[
\vdots
\]
\(([qa \ldots a], \rightarrow, a, p, [qa \ldots aa], \rightarrow, \epsilon)\)

\([qa \ldots aa]\) larger than \(F(|G|)\) \Rightarrow not one-way definable
Running the Algorithm on Examples (I)

Reverse

\[ \tau_I(p) = q \]

\[ q(a) \xrightarrow{p} qa \]
\[ q(b) \xrightarrow{p} qb \]
\[ q(\triangle) \xrightarrow{} \epsilon \]

\[
(s_I, \rightarrow, \triangleright, p, [q], \rightarrow, \epsilon)

([q], \rightarrow, a, p, [qa], \rightarrow, \epsilon)

([qa], \rightarrow, a, p, [qaa], \rightarrow, \epsilon)

\ldots

([qa \ldots a], \rightarrow, a, p, [qa \ldots aa], \rightarrow, \epsilon)

[qa \ldots aa] larger than \( F(|\mathcal{G}|) \) \( \Rightarrow \) not one-way definable
Running the Algorithm on Examples (I)

Reverse

\[ \tau_1(p) = q \]

\[ q(a) \xrightarrow{p} qa \]
\[ q(b) \xrightarrow{p} qb \]
\[ q(\triangle) \xrightarrow{} \epsilon \]

\( (s_I, \rightarrow, \triangle, p, [q], \rightarrow, \epsilon) \)
\( ([q], \rightarrow, a, p, [qa], \rightarrow, \epsilon) \)
\( ([qa], \rightarrow, a, p, [qaa], \rightarrow, \epsilon) \)

\[ \cdots \]
\[ ([qa \ldots a], \rightarrow, a, p, [qa \ldots aa], \rightarrow, \epsilon) \]

\([qa \ldots aa]\) larger than \( F(|G|) \) ⇒ not one-way definable
Running the Algorithm on Examples (I)

Reverse

\[ \tau_1(p) = q \]

\[
\begin{align*}
q(a) & \xrightarrow{p} qa \\
q(b) & \xrightarrow{p} qb \\
q(\triangle) & \rightarrow \epsilon
\end{align*}
\]

\[
\begin{align*}
(s_I, \rightarrow, \triangledown, p, [q], \rightarrow, \epsilon) \\
([q], \rightarrow, a, p, [qa], \rightarrow, \epsilon) \\
([qa], \rightarrow, a, p, [qaa], \rightarrow, \epsilon) \\
\ldots \\
([qa \ldots a], \rightarrow, a, p, [qa \ldots aa], \rightarrow, \epsilon)
\end{align*}
\]

\[ [qa \ldots aa] \text{ larger than } F(|\mathcal{G}|) \Rightarrow \text{not one-way definable} \]
Running the Algorithm on Examples (II)

LastFirst: $\triangleright\{a, b\}^+\triangleleft \rightarrow \triangleright\{a, b, c\}^+\triangleleft$ with $\sigma \in \{a, b\}$

$\tau_I(p_a) = \triangleright a \ q\triangleleft$

$\tau_I(p_b) = \triangleright b \ q\triangleleft$

$q(a) \xrightarrow{p_a} q \ c$  
$q(b) \xrightarrow{p_a} q \ c$

$q(a) \xrightarrow{p_b} q \ c$  
$q(b) \xrightarrow{p_b} q \ c$

$q(a) \xrightarrow{p'} \epsilon$  
$q(b) \xrightarrow{p'} \epsilon$

$[q]_{p_a}, [q]_{p_b}$ are periodic of period $c$

Push($q, p_a, c$) = $cq$

Push($q, p_b, c$) = $cq$

$(s_I, \rightarrow, \triangleright, p_a, [q\triangleleft], \rightarrow, \triangleright a)$

using $q(a) \xrightarrow{p_a} q \ c$ and pushing, $[qc\triangleleft]$ becomes $c[q\triangleleft]$

$([q\triangleleft], \rightarrow, a, p_a, [q\triangleleft], \rightarrow, c)$  
$([q\triangleleft], \rightarrow, b, p_a, [q\triangleleft], \rightarrow, c)$

$([q\triangleleft], \rightarrow, a, p_b, [q\triangleleft], \rightarrow, c)$  
$([q\triangleleft], \rightarrow, b, p_b, [q\triangleleft], \rightarrow, c)$

$([q\triangleleft], \rightarrow, a, p', s_F, \rightarrow, \triangleleft)$  
$([q\triangleleft], \rightarrow, b, p', s_F, \rightarrow, \triangleleft)$
Running the Algorithm on Examples (II)

LastFirst: \[ \triangleright \{a, b\}^+ \triangleright \rightarrow \triangleright \{a, b, c\}^+ \triangleright \] with \( \sigma \in \{a, b\} \)

\[
\begin{align*}
\tau_I(p_a) &= \triangleright a q \triangleright \\
\tau_I(p_b) &= \triangleright b q \triangleright
\end{align*}
\]

\[
\begin{align*}
q(a) &\xrightarrow{p_a} q c \\
q(b) &\xrightarrow{p_a} q c \\
q(a) &\xrightarrow{p_b} q c \\
q(b) &\xrightarrow{p_b} q c \\
q(a) &\xrightarrow{p'} \epsilon \\
q(b) &\xrightarrow{p'} \epsilon
\end{align*}
\]

\([q]_{p_a}, [q]_{p_b}\) are periodic of period \(c\) \[
\left\{
\begin{array}{l}
\text{Push}(q, p_a, c) = cq \\
\text{Push}(q, p_b, c) = cq
\end{array}
\right.
\]

\((s_I, \rightarrow, \triangleright, p_a, [q\triangleright], \rightarrow, \triangleright a)\) using \(q(a) \xrightarrow{p_a} q c\) and pushing, \([qc\triangleright]\) becomes \(c[q\triangleright]\)

\(([q\triangleright], \rightarrow, a, p_a, [q\triangleright], \rightarrow, c) \quad ([q\triangleright], \rightarrow, b, p_a, [q\triangleright], \rightarrow, c)\)

\(([q\triangleright], \rightarrow, a, p_b, [q\triangleright], \rightarrow, c) \quad ([q\triangleright], \rightarrow, b, p_b, [q\triangleright], \rightarrow, c)\)

\(([q\triangleright], \rightarrow, a, p', s_F, \rightarrow, \triangleright) \quad ([q\triangleright], \rightarrow, b, p', s_F, \rightarrow, \triangleright)\)
Running the Algorithm on Examples (II)

LastFirst: \( \{a, b\}^+ \) → \( \{a, b, c\}^+ \) with \( \sigma \in \{a, b\} \)

\[\tau_I(p_a) = \triangleright a \ q \triangleleft \]
\[\tau_I(p_b) = \triangleright b \ q \triangleleft \]

\[q(a) \overset{p_a}{\rightarrow} q \ c \quad q(b) \overset{p_a}{\rightarrow} q \ c\]
\[q(a) \overset{p_b}{\rightarrow} q \ c \quad q(b) \overset{p_b}{\rightarrow} q \ c\]
\[q(a) \overset{p'}{\rightarrow} \epsilon \quad q(b) \overset{p'}{\rightarrow} \epsilon\]

\([q]_{p_a}, [q]_{p_b}\) are periodic of period \(c\)

\[\{\text{Push}(q, p_a, c) = cq, \text{Push}(q, p_b, c) = cq\}\]

\((s_I, \rightarrow, \triangleright, p_a, [q\triangleleft], \rightarrow, \triangleright a)\)

using \(q(a) \overset{p_a}{\rightarrow} q \ c\) and pushing, \([qc\triangleleft]\) becomes \(c[q\triangleleft]\)

\([q\triangleleft], \rightarrow, a, p_a, [q\triangleleft], \rightarrow, c\)  \([q\triangleleft], \rightarrow, b, p_a, [q\triangleleft], \rightarrow, c\)

\([q\triangleleft], \rightarrow, a, p_b, [q\triangleleft], \rightarrow, c\)  \([q\triangleleft], \rightarrow, b, p_b, [q\triangleleft], \rightarrow, c\)

\([q\triangleleft], \rightarrow, a, p', s_F, \rightarrow, \triangleleft\)  \([q\triangleleft], \rightarrow, b, p', s_F, \rightarrow, \triangleleft\)
Running the Algorithm on Examples (II)

LastFirst : \(\triangleright\{a, b\}^+\triangleleft\) \(\rightarrow\) \(\triangleright\{a, b, c\}^+\triangleleft\) with \(\sigma \in \{a, b\}\)

\(\tau_I(p_a) = \triangleright a\ q\triangleleft\)

\(\tau_I(p_b) = \triangleright b\ q\triangleleft\)

\(q(a) \overset{p_a}{\rightarrow} q\ c\) \(\quad q(b) \overset{p_a}{\rightarrow} q\ c\)

\(q(a) \overset{p_b}{\rightarrow} q\ c\) \(\quad q(b) \overset{p_b}{\rightarrow} q\ c\)

\(q(a) \overset{p'}{\rightarrow} \epsilon\) \(\quad q(b) \overset{p'}{\rightarrow} \epsilon\)

\([q]_{p_a}, [q]_{p_b}\) are periodic of period \(c\) \(\left\{\begin{array}{l}
\text{Push}(q, p_a, c) = cq \\
\text{Push}(q, p_b, c) = cq
\end{array}\right.\)

\((s_I, \rightarrow, \triangleright, p_a, [q\triangleleft], \rightarrow, \triangleright a)\)

using \(q(a) \overset{p_a}{\rightarrow} q\ c\) and pushing, \([qc\triangleleft]\) becomes \(c[q\triangleleft]\)

\(([q\triangleleft], \rightarrow, a, p_a, [q\triangleleft], \rightarrow, c)\) \(\quad ([q\triangleleft], \rightarrow, b, p_a, [q\triangleleft], \rightarrow, c)\)

\(([q\triangleleft], \rightarrow, a, p_b, [q\triangleleft], \rightarrow, c)\) \(\quad ([q\triangleleft], \rightarrow, b, p_b, [q\triangleleft], \rightarrow, c)\)

\(([q\triangleleft], \rightarrow, a, p', s_F, \rightarrow, \triangleleft)\) \(\quad ([q\triangleleft], \rightarrow, b, p', s_F, \rightarrow, \triangleleft)\)
To Sum up

If successful, $A$ is of size at most 4-exponential in the size of the input D2FST.

Guessing a failure requires only 3-exponential space.

Next ?

- On going : the second step of normalization is useless $\Rightarrow$ drop one exponential
- Educated guess : pushing only powers of (complete) periods through periodic language
- (False ?) hope : technics could be adapted to (normalized) functional 2FST.
To Sum up

If successful, $A$ is of size at most 4-exponential in the size of the input D2FST.

Guessing a failure requires only 3-exponential space.

Next?

- On going: the second step of normalization is useless $\Rightarrow$ drop one exponential
- Educated guess: pushing only powers of (complete) periods through periodic language
- (False?) hope: technics could be adapted to (normalized) functional 2FST.
More Example

\[
[A] (\triangleright u_1 \# \sigma u_2 \triangleright) = \triangleright a | u_1 | + | u_2 | + 2 \triangleleft \text{ if } \sigma = a, \text{ and }
\]
\[
[A] (\triangleright u_1 \# \sigma u_2 \triangleright) = \triangleright ba | u_1 | \triangleleft \text{ if } \sigma = b
\]
The look-ahead automaton

More Example
More Example

\[(s_I, \rightarrow, \triangleright, T_a, (1, 4)\triangleleft T_a, \rightarrow, \triangleright a)\]
\[(s_I, \rightarrow, \triangleright, T_b, (1, 4)\triangleleft T_b, \rightarrow, \triangleright b)\]

\[((1, 4)\triangleleft T_a, \rightarrow, \sigma, T_a, (1, 4)\triangleleft T_a, \rightarrow, c)\] for \(\sigma \in \{a, b\}\)
\[((1, 4)\triangleleft T_b, \rightarrow, \sigma, T_b, (1, 4)\triangleleft T_b, \rightarrow, c)\] for \(\sigma \in \{a, b\}\)
\[((1, 4)\triangleleft T_a, \rightarrow, a, T_{\triangleleft}, \epsilon, \rightarrow, \triangle)\)
\[((1, 4)\triangleleft T_b, \rightarrow, b, T_{\triangleleft}, \epsilon, \rightarrow, \triangle)\)