Jardin et al. 2014

LIN 629, Fall 2022

Adil Soubki Salam Khalifa

Introduction

1. They present the Structured Onward Subsequential Function Inference Algorithm (SOSFIA) which can identify some classes of sequential functions in linear time and data.
2. The main way they do this is by using *a priori* knowledge of structure shared by functions in the class (No state merging!).
3. One of these classes is are the Input Strictly Local functions.

Preliminaries

Prefixes

The prefixes of $w \in \Sigma^*$ are,

$$\mathtt{pref}(w) = \{u \in \Sigma^* \mid (\exists v)[uv = w]\}$$

Preliminaries

Prefixes

The prefixes of $w \in \Sigma^*$ are,

$$\mathtt{pref}(w) = \{u \in \Sigma^* \mid (\exists v)[uv = w]\}$$

This can be extended to a string set, $S \subseteq \Sigma^*$, where,

$$\mathtt{pref}(S) = \bigcup_{w \in S} \mathtt{pref}(w)$$

Preliminaries

Prefixes

The prefixes of $w \in \Sigma^*$ are,

$$\texttt{pref}(w) = \{u \in \Sigma^* \mid (\exists v)[uv = w]\}$$

This can be extended to a string set, $S \subseteq \Sigma^*$, where,

$$\texttt{pref}(S) = \bigcup_{w \in S} \texttt{pref}(w)$$

Examples

$$\texttt{pref}(\lambda) = \{\lambda\}$$
$$\texttt{pref}(\mathsf{aba}) = \{\lambda, \mathsf{a}, \mathsf{ab}, \mathsf{aba}\}$$
$$\texttt{pref}(\{\mathsf{aba}, \mathsf{bb}\}) = \{\lambda, \mathsf{a}, \mathsf{ab}, \mathsf{abab}, \mathsf{bb}\}$$

Preliminaries

Shared Prefixes

The *shared prefixes* of S are the prefixes shared by all strings in S.

$$\mathtt{sh\_pref}(S) = \bigcap_{w \in S} \mathtt{pref}(w)$$

Examples

$$\begin{aligned}
\mathtt{sh_pref}(\{\ \}) &= \{\ \} \\
\mathtt{sh_pref}(\{\mathsf{aba}, \mathsf{bb}\}) &= \{\lambda, \mathsf{a}, \mathsf{ab}, \mathsf{aba}\} \cap \{\lambda, \mathsf{b}, \mathsf{bb}\} \\
&= \{\lambda\} \\
\mathtt{sh_pref}(\{\mathsf{aba}, \mathsf{a}\}) &= \{\lambda, \mathsf{a}, \mathsf{ab}, \mathsf{aba}\} \cap \{\lambda, \mathsf{a}\} \\
&= \{\lambda, \mathsf{a}\}
\end{aligned}$$

Preliminaries

Longest Common Prefix

The longest common prefix ($\texttt{lcp}$) of S is then,

$$\texttt{lcp}(S) = \{w \in \texttt{sh\_pref}(S) \mid \forall v \in \texttt{sh\_pref}(S) : |v| \leq |w|\}$$

We define the $\texttt{lcp}$ of the empty set to be λ.

Examples

$$\texttt{lcp}(\{\ \}) = \lambda$$
$$\texttt{lcp}(\{\mathsf{aba}, \mathsf{a}\}) = \mathsf{a}$$

Preliminaries

Relations

Given an input alphabet Σ and an output alphabet Δ, a relation from Σ to Δ is a subset of $\Sigma^* \times \Delta^*$.

Preliminaries

Relations

Given an input alphabet Σ and an output alphabet Δ, a relation from Σ to Δ is a subset of $\Sigma^* \times \Delta^*$.

Functions

We call $t : \Sigma^* \rightarrow \Delta^*$ a function iff $\forall w \in \Sigma^*$,

$$(w, v), (w, v') \in t \implies v = v'$$

Preliminaries

Relations

Given an input alphabet Σ and an output alphabet Δ, a relation from Σ to Δ is a subset of $\Sigma^* \times \Delta^*$.

Functions

We call $t : \Sigma^* \rightarrow \Delta^*$ a function iff $\forall w \in \Sigma^*$,

$$(w, v), (w, v') \in t \implies v = v'$$

Totality

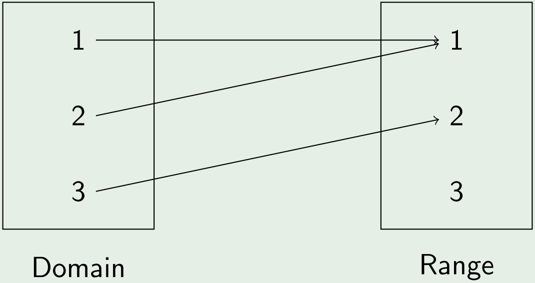
A function $t : \Sigma^* \rightarrow \Delta^*$ is total iff,

$$\forall w \in \Sigma^*, \exists v \in \Delta^* \text{ s.t. } (w, v) \in t$$

Otherwise, the function is partial.

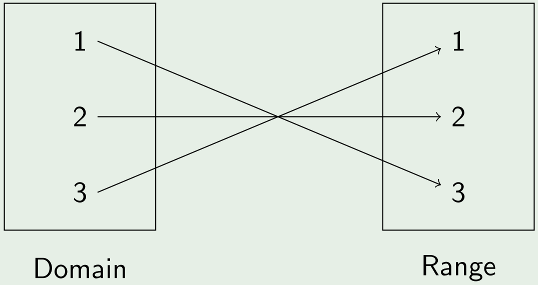
Preliminaries

Examples



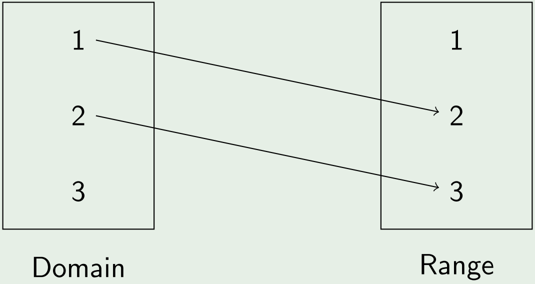
Preliminaries

Examples



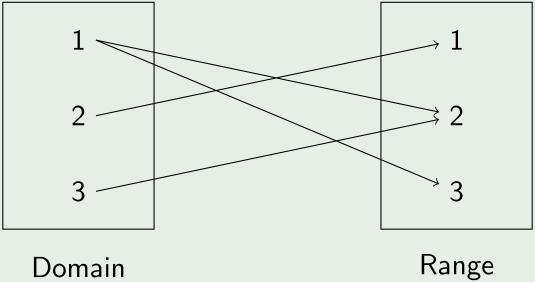
Preliminaries

Examples



Preliminaries

Examples



Preliminaries

Tails of a Function

For any function $t : \Sigma^* \rightarrow \Delta^*$ and $w \in \Sigma^*$, we define the tails of w with respect to t as,

$$\texttt{tails}_t(w) = \{(x, v) \mid t(wx) = uv \wedge u = \texttt{lcp}(t(w\Sigma^*))\}$$

Preliminaries

Tails of a Function

For any function $t : \Sigma^* \to \Delta^*$ and $w \in \Sigma^*$, we define the tails of w with respect to t as,

$$\texttt{tails}_t(w) = \{(x, v) \mid t(wx) = uv \wedge (u = \texttt{lcp}(t(w\Sigma^*)))\}$$

Preliminaries

Tails of a Function

For any function $t : \Sigma^* \rightarrow \Delta^*$ and $w \in \Sigma^*$, we define the tails of w with respect to t as,

$$\texttt{tails}_t(w) = \{(x, v) \mid t(wx) = uv \wedge (u = \texttt{lcp}(t(w\Sigma^*)))\}$$

Examples

Let $\Sigma = \Delta = \{\mathrm{a}, \mathrm{b}, \mathrm{p}\}$ and t be the identity function.

$$\begin{aligned}\texttt{tails}_t(\mathrm{ba}) &= \{(\mathrm{a}, \mathrm{a}), (\mathrm{b}, \mathrm{b}), (\mathrm{p}, \mathrm{p}), (\mathrm{ba}, \mathrm{ba}), \cdots\} \\ \texttt{tails}_t(\mathrm{bab}) &= \{(\mathrm{a}, \mathrm{a}), (\mathrm{b}, \mathrm{b}), (\mathrm{p}, \mathrm{p}), (\mathrm{ba}, \mathrm{ba}), \cdots\}\end{aligned}$$

Note that $\texttt{tails}_t(\mathrm{ba}) = \texttt{tails}_t(\mathrm{bab})$.

Preliminaries

Tails of a Function

For any function $t : \Sigma^* \to \Delta^*$ and $w \in \Sigma^*$, we define the tails of w with respect to t as,

$$\texttt{tails}_t(w) = \{(x, v) \mid t(wx) = uv \wedge (u = \texttt{lcp}(t(w\Sigma^*)))\}$$

Examples

Let $\Sigma = \Delta = \{\text{a}, \text{b}, \text{p}\}$ and t require a "b" to follow every "a".

$$\texttt{tails}_t(\text{ba}) = \{(\text{a}, \text{b}), (\text{b}, \text{b}), (\text{p}, \text{b}), (\text{ba}, \text{ba}), \cdots\}$$
$$\texttt{tails}_t(\text{bab}) = \{(\text{a}, \text{a}), (\text{b}, \text{b}), (\text{p}, \text{p}), (\text{ba}, \text{ba}), \cdots\}$$

Note that $\texttt{tails}_t(\text{ba}) \neq \texttt{tails}_t(\text{bab})$.

Preliminaries

Definition

Two strings $w, w' \in \Sigma^*$ are tail-equivalent with respect to a function t iff,

$$\mathtt{tails}_t(w) = \mathtt{tails}_t(w')$$

which we denote as $w \sim_t w'$.

Preliminaries

Definition

Two strings $w, w' \in \Sigma^*$ are tail-equivalent with respect to a function t iff,

$$\texttt{tails}_t(w) = \texttt{tails}_t(w')$$

which we denote as $w \sim_t w'$.

Remark

The equivalence relation $\sim_t$ partitions Σ^*.

Preliminaries

Definition

Two strings $w, w' \in \Sigma^*$ are tail-equivalent with respect to a function t iff,

$$\texttt{tails}_t(w) = \texttt{tails}_t(w')$$

which we denote as $w \sim_t w'$.

Remark

The equivalence relation $\sim_t$ partitions Σ^*.

Examples

Let $\Sigma = \Delta = \{\mathrm{a}, \mathrm{b}, \mathrm{p}\}$ and t be the identity function.

$$\texttt{tails}_t(\mathrm{ba}) = \{(\mathrm{a}, \mathrm{a}), (\mathrm{b}, \mathrm{b}), (\mathrm{p}, \mathrm{p}), (\mathrm{ba}, \mathrm{ba}), \cdots\}$$
$$\texttt{tails}_t(\mathrm{bab}) = \{(\mathrm{a}, \mathrm{a}), (\mathrm{b}, \mathrm{b}), (\mathrm{p}, \mathrm{p}), (\mathrm{ba}, \mathrm{ba}), \cdots\}$$

How many blocks does $\sim_t$ partition Σ^* into?

Preliminaries

Definition

Two strings $w, w' \in \Sigma^*$ are tail-equivalent with respect to a function t iff,

$$\texttt{tails}_t(w) = \texttt{tails}_t(w')$$

which we denote as $w \sim_t w'$.

Remark

The equivalence relation $\sim_t$ partitions Σ^*.

Examples

Let $\Sigma = \Delta = \{\mathrm{a}, \mathrm{b}, \mathrm{p}\}$ and t require a “b” to follow every “a”.

$$\texttt{tails}_t(\mathrm{ba}) = \{(\mathrm{a}, \mathrm{b}), (\mathrm{b}, \mathrm{b}), (\mathrm{p}, \mathrm{b}), (\mathrm{ba}, \mathrm{ba}), \cdots\}$$
$$\texttt{tails}_t(\mathrm{bab}) = \{(\mathrm{a}, \mathrm{a}), (\mathrm{b}, \mathrm{b}), (\mathrm{p}, \mathrm{p}), (\mathrm{ba}, \mathrm{ba}), \cdots\}$$

How many blocks does $\sim_t$ partition Σ^* into?

Preliminaries

Subsequential Functions

A function t is subsequential iff $\sim_t$ partitions Σ^* into *finitely* many blocks.

Preliminaries

Subsequential Functions

A function t is subsequential iff $\sim_t$ partitions Σ^* into *finitely* many blocks.

Remark

I think what this is getting at is that subsequential functions "naturally" correspond to deterministic finite-state transducers and vice versa.

Preliminaries

Subsequential Functions

A function t is subsequential iff $\sim_t$ partitions Σ^* into *finitely* many blocks.

Remark

I think what this is getting at is that subsequential functions "naturally" correspond to deterministic finite-state transducers and vice versa.

Examples

Let $\Sigma = \Delta = \{\text{a}, \text{b}, \text{p}\}$ and t be the identity function.

Is t a subsequential function?

Let $\Sigma = \Delta = \{\text{a}, \text{b}, \text{p}\}$ and f require a "b" to follow every "a".

Is f a subsequential function?

Traditional Subsequential Transducers

Definition

A subsequential finite-state transducer (SFST) is a 6-tuple $\langle Q, q_0, \Sigma, \Delta, \delta, p \rangle$ where,

1. Q is a *finite* set of states.
2. $q_0 \in Q$ is the initial state.
3. Σ is a *finite* input alphabet.
4. Δ is a *finite* output alphabet.
5. $\delta \subseteq Q \times \Sigma \times \Delta^* \times Q$ is the transition function.
6. $p : Q \to \Delta^*$ is the output function.

Additionally, the transition function must be deterministic. That is,

$$(q, \sigma, w, r), (q, \sigma, v, s) \in \delta \implies (r = s) \wedge (w = v)$$

Traditional Subsequential Transducers

Theorem

If we denote the relation described by a SFST τ as,

$$R(\tau) = \{(w, vv') \mid (\exists q)[(q_0, w, v, q) \in \delta^* \wedge p(q) = v']\}$$

Then for every for every subsequential function t, there is a subsequential transducer τ which computes it ($R(\tau) = t$), and likewise for every subsequential transducer τ, $R(\tau)$ is a subsequential function.

Delimited Subsequential Transducers

Definition

A delimited subsequential finite-state transducer (DSFST) is a 6-tuple $\langle Q, q_0, q_f, \Sigma, \Delta, \delta \rangle$ where,

1. Q is a *finite* set of states.
2. $q_0 \in Q$ is the initial state.
3. $q_f \in Q$ is the final state.
4. Σ is a *finite* input alphabet.
5. Δ is a *finite* output alphabet.
6. $\delta \subseteq Q \times \Sigma \cup \{\ltimes, \rtimes\} \times \Delta^* \times Q$ is the transition function.

Where $\ltimes, \rtimes \notin \Sigma$ and the following holds,

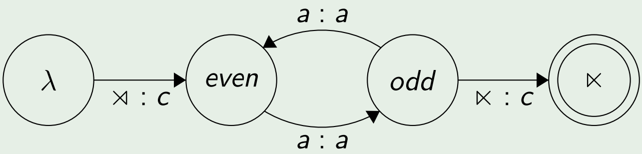
1. $(q, \sigma, u, q') \in \delta \implies (q \neq q_f) \wedge (q' \neq q_0)$,
2. $(q, \sigma, u, q_f) \in \delta \implies (\sigma = \ltimes) \wedge (q \neq q_0)$,
3. $[(q_0, \sigma, u, q') \in \delta \implies \sigma = \rtimes] \wedge [(q_0, \rtimes, u, q') \implies q = q_0]$
4. $(q, \sigma, w, r), (q, \sigma, v, s) \in \delta \implies (r = s) \wedge (w = v)$

Reachable States

Definition

A state q in a DSFST τ is reachable iff there exists $w \in \Sigma^*, v \in \Delta^*$ such that $(q_0, \rtimes w, v, q) \in \delta^*$.

Examples

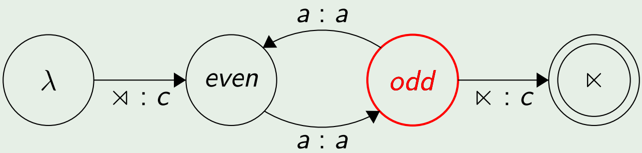


Reachable States

Definition

A state q in a DSFST τ is reachable iff there exists $w \in \Sigma^*, v \in \Delta^*$ such that $(q_0, \rtimes w, v, q) \in \delta^*$.

Examples

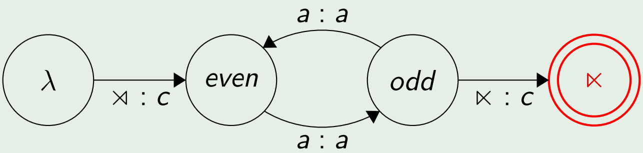


Reachable States

Definition

A state q in a DSFST τ is reachable iff there exists $w \in \Sigma^*, v \in \Delta^*$ such that $(q_0, \rtimes w, v, q) \in \delta^*$.

Examples

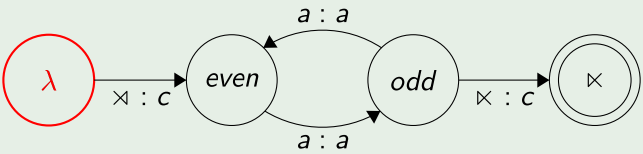


Reachable States

Definition

A state q in a DSFST τ is reachable iff there exists $w \in \Sigma^*, v \in \Delta^*$ such that $(q_0, \rtimes w, v, q) \in \delta^*$.

Examples

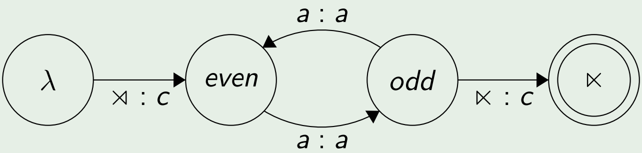


Useful States

Definition

A state q in a DSFST τ is useful iff q is *reachable* and there exists $w \in \Sigma^*, v \in \Delta^*$ such that $(q, w\ltimes, v, q_f) \in \delta^*$.

Examples

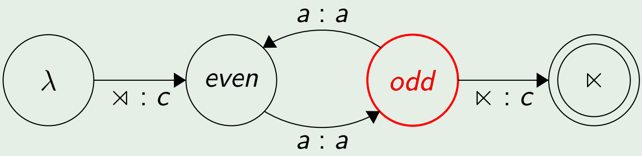


Useful States

Definition

A state q in a DSFST τ is useful iff q is *reachable* and there exists $w \in \Sigma^*, v \in \Delta^*$ such that $(q, w\ltimes, v, q_f) \in \delta^*$.

Examples

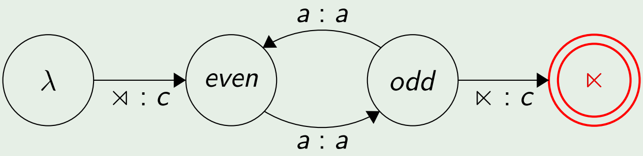


Useful States

Definition

A state q in a DSFST τ is useful iff q is *reachable* and there exists $w \in \Sigma^*, v \in \Delta^*$ such that $(q, w\ltimes, v, q_f) \in \delta^*$.

Examples

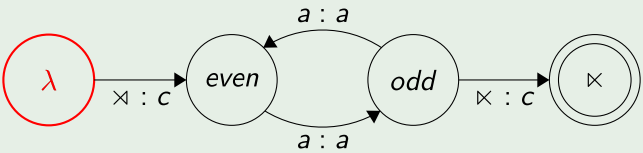


Useful States

Definition

A state q in a DSFST τ is useful iff q is *reachable* and there exists $w \in \Sigma^*, v \in \Delta^*$ such that $(q, w\ltimes, v, q_f) \in \delta^*$.

Examples

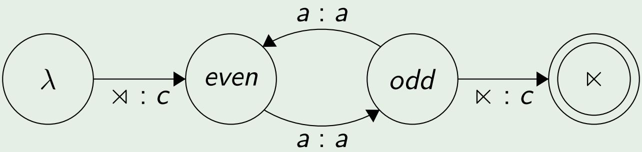


Trimmed Transducers

Definition

A transducer is trimmed iff every state other than q_0 and q_f is *useful*.

Examples

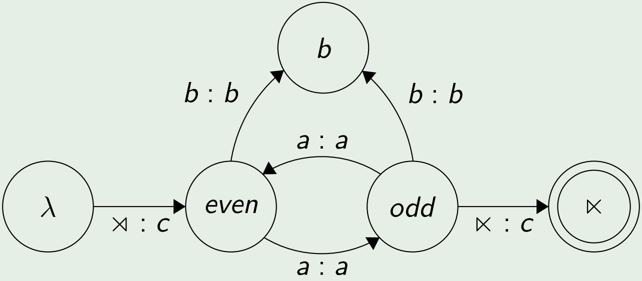


Trimmed Transducers

Definition

A transducer is trimmed iff every state other than q_0 and q_f is *useful*.

Examples



Onward Transducers

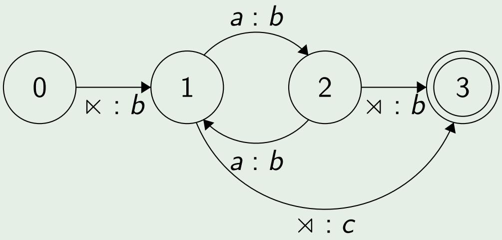
Definition

A DSFST τ is onward iff the outgoing transitions of each noninitial state share no nonempty prefix.

$$(\forall q \in (Q - q_0))[\texttt{lcp}\{w \in \Sigma^* \mid (\exists a \in \Sigma, r \in Q)[(q, a, w, r) \in \delta]\} = \lambda]$$

Intuitively, a transducer is *onward* iff there is no delay in writing the output strings. As the input symbols are consumed, the output is written the moment it is determined.

Examples



Onward Transducers

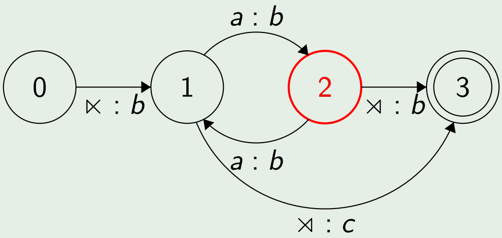
Definition

A DSFST τ is onward iff the outgoing transitions of each noninitial state share no nonempty prefix.

$$(\forall q \in (Q - q_0))[\mathtt{lcp}\{w \in \Sigma^* \mid (\exists a \in \Sigma, r \in Q)[(q, a, w, r) \in \delta]\} = \lambda]$$

Intuitively, a transducer is *onward* iff there is no delay in writing the output strings. As the input symbols are consumed, the output is written the moment it is determined.

Examples



Onward Transducers

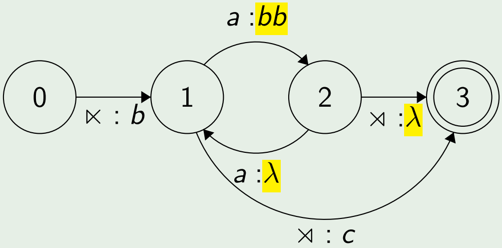
Definition

A DSFST τ is onward iff the outgoing transitions of each noninitial state share no nonempty prefix.

$$(\forall q \in (Q - q_0))[\mathtt{lcp}\{w \in \Sigma^* \mid (\exists a \in \Sigma, r \in Q)[(q, a, w, r) \in \delta]\} = \lambda]$$

Intuitively, a transducer is *onward* iff there is no delay in writing the output strings. As the input symbols are consumed, the output is written the moment it is determined.

Examples



Deriving Onward Transducers

Theorem

For any trimmed DSFST τ, there is an onward trimmed DSFST τ' such that $R(\tau) = R(\tau')$

Remarks

What this is saying is that for any DSFST τ, there is an *onward DSFST* with the exact same structure which recognizes the same function described by τ.

For the purposes of this presentation I state this without proof. The important thing is that there is some canonical onward DSFST for every DSFST and additionally there is a constructive way to find it.

Learning Paradigm

The results for the algorithm presented in this paper (SOSFIA) are demonstrated in the identification in the limit learning paradigm.

Definition

A class $\mathbb{T}$ of functions is represented by a class $\mathbb{R}$ of representations if every $r \in \mathbb{R}$ is of finite size and there is a function $\mathcal{L} : \mathbb{R} \rightarrow \mathbb{T}$ which is both total and surjective

Learning Paradigm

The results for the algorithm presented in this paper (SOSFIA) are demonstrated in the identification in the limit learning paradigm.

Definition

A class $\mathbb{T}$ of functions is represented by a class $\mathbb{R}$ of representations if every $r \in \mathbb{R}$ is of finite size and there is a function $\mathcal{L} : \mathbb{R} \rightarrow \mathbb{T}$ which is both total and surjective (i.e. each element of the range is mapped to by *at least one element* of the domain).

Learning Paradigm

The results for the algorithm presented in this paper (SOSFIA) are demonstrated in the identification in the limit learning paradigm.

Definition

A class $\mathbb{T}$ of functions is represented by a class $\mathbb{R}$ of representations if every $r \in \mathbb{R}$ is of finite size and there is a function $\mathcal{L} : \mathbb{R} \to \mathbb{T}$ which is both total and surjective (i.e. each element of the range is mapped to by *at least one element* of the domain).

Remark

The class of subsequential functions can be represented by the class of DSFSTs.

Learning Paradigm

Definition

Let $\mathbb{T}$ be a class of functions represented by some class $\mathbb{R}$ of representations.

1. A sample S for a function $t \in \mathbb{T}$ is a finite set of data consistent with t, that is to say,
$$(w, v) \in S \iff t(w) = v$$
2. The size of a sample S is the sum of the length of the strings it is composed of.
$$|S| = \sum_{(w,v)\in S} |w| + |v|$$
3. A $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$ is a program that takes as input a sample for a function $t \in \mathbb{T}$ and outputs a representation from $\mathbb{R}$.

Learning Paradigm

Definition (Characteristic Sample)

For a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$, a sample CS is a characteristic sample of a function $t \in \mathbb{T}$ if for all samples S such that $CS \subseteq S$, $\mathfrak{A}$ returns a representation r such that, for all $w \in \mathtt{dom}(t)$, $r(w) = t(w)$.

Definition (Strong Characteristic Sample)

For a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$, a sample CS is a strong characteristic sample of a representation $r \in \mathbb{R}$ if for all samples S for $\mathcal{L}(r)$ such that $CS \subseteq S$, $\mathfrak{A}$ returns r.

In English, this is saying that a sample is a characteristic sample if adding more elements does not (loosely speaking) change the output for $\mathfrak{A}$.

Learning Paradigm

Definition (Characteristic Sample)

For a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$, a sample CS is a characteristic sample of a function $t \in \mathbb{T}$ if for all samples S such that $CS \subseteq S$, $\mathfrak{A}$ returns a representation r such that, for all $w \in \mathtt{dom}(t)$, $r(w) = t(w)$.

Definition (Strong Characteristic Sample)

For a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$, a sample CS is a strong characteristic sample of a representation $r \in \mathbb{R}$ if for all samples S for $\mathcal{L}(r)$ such that $CS \subseteq S$, $\mathfrak{A}$ returns r.

In English, this is saying that a sample is a characteristic sample if adding more elements does not (loosely speaking) change the output for $\mathfrak{A}$.

Learning Paradigm

Definition (Characteristic Sample)

For a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$, a sample CS is a characteristic sample of a function $t \in \mathbb{T}$ if for all samples S such that $CS \subseteq S$, $\mathfrak{A}$ returns a representation r such that, for all $w \in \texttt{dom}(t)$, $r(w) = t(w)$.

Definition (Strong Characteristic Sample)

For a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$, a sample CS is a strong characteristic sample of a representation $r \in \mathbb{R}$ if for all samples S for $\mathcal{L}(r)$ such that $CS \subseteq S$, $\mathfrak{A}$ returns r.

In English, this is saying that a sample is a characteristic sample if adding more elements does not (loosely speaking) change the output for $\mathfrak{A}$.

Learning Paradigm

Definition (Characteristic Sample)

For a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$, a sample CS is a characteristic sample of a function $t \in \mathbb{T}$ if for all samples S such that $CS \subseteq S$, $\mathfrak{A}$ returns a representation r such that, for all $w \in \mathtt{dom}(t)$, $r(w) = t(w)$.

Definition (Strong Characteristic Sample)

For a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$, a sample CS is a strong characteristic sample of a representation $r \in \mathbb{R}$ if for all samples S for $\mathcal{L}(r)$ such that $CS \subseteq S$, $\mathfrak{A}$ returns r.

In English, this is saying that a sample is a characteristic sample if adding more elements does not (loosely speaking) change the output for $\mathfrak{A}$.

Learning Paradigm

Definition (Strong Characteristic Sample)

For a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$, a sample CS is a strong characteristic sample of a representation $r \in \mathbb{R}$ if for all samples S for $\mathcal{L}(r)$ such that $CS \subseteq S$, $\mathfrak{A}$ returns r.

This is the definition Jardine et al. 2014 uses so we will stick to that.

Output-Empty Transducers

Definition

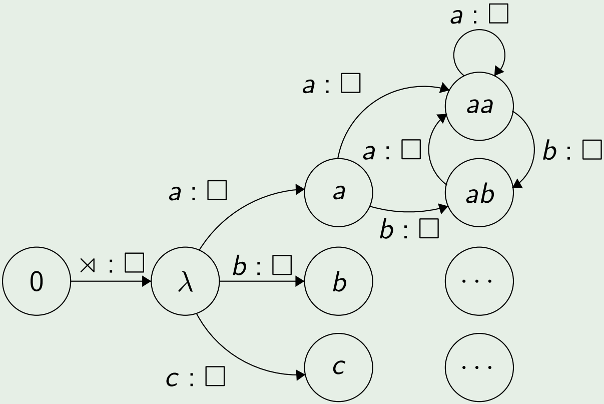
An output-empty transducer $\tau_{\Box}$ is a DSFST $\langle Q, q_0, q_f, \Sigma, \{\Box\}, \delta \rangle$ such that for all $(q, a, u, q') \in \delta$, $u = \Box$.

An output-empty transducer τ defines a class of functions $\mathcal{T}$ which is exactly the set of functions which can be created by taking the states and transitions of τ and replacing the blanks with output strings, maintaining onwardness.

The goal is to create an algorithm that can fill in these blanks.

Output-Empty Transducers

Example



The Inference Algorithm

The algorithm takes an output-empty transducer $\tau_{\Box}$, and a finite sample S from some target function $t \in \mathcal{T}_{\tau_{\Box}}$.
At each state, it sets the output of each outgoing transition to be *the minimal change in the output* generated by this transition according to S.

The Inference Algorithm

The algorithm takes an output-empty transducer $\tau_\Box$, and a finite sample S from some target function $t \in \mathcal{T}_{\tau_\Box}$.
At each state, it sets the output of each outgoing transition to be *the minimal change in the output* generated by this transition according to S.

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the `lcp` of all $t(wv)$ that are in S.

$$\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

The Inference Algorithm

The algorithm takes an output-empty transducer $\tau_{\square}$, and a finite sample S from some target function $t \in \mathcal{T}_{\tau_{\square}}$.
At each state, it sets the output of each outgoing transition to be *the minimal change in the output* generated by this transition according to S.

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the `lcp` of all $t(wv)$ that are in S.

$$\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

common_out

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the lcp of all $t(wv)$ that are in S.

$$\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$\texttt{common\_out}_S(a) = ???$

common_out

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the `lcp` of all $t(wv)$ that are in S.

$$\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{ll} (\mathbf{a}npa, ama), & (\mathbf{a}npo, amo) \\ (\mathbf{a}na, ana), & (\mathbf{a}no, ano) \\ (\mathbf{a}nda, anda), & (\mathbf{a}ndo, ando) \end{array} \right\}$$

$\texttt{common\_out}_S(a) = ???$

common_out

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the lcp of all $t(wv)$ that are in S.

$$\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{ll} (\text{a}npa, \textbf{ama}), & (\text{a}npo, \textbf{amo}) \\ (\text{a}na, \textbf{ana}), & (\text{a}no, \textbf{ano}) \\ (\text{a}nda, \textbf{anda}), & (\text{a}ndo, \textbf{ando}) \end{array} \right\}$$

$\texttt{common\_out}_S(a) = ???$

common_out

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the $\mathtt{lcp}$ of all $t(wv)$ that are in S.

$$\mathtt{common\_out}_S(w) = \mathtt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{rr} (anpa, \textbf{ama}), & (anpo, \textbf{amo}) \\ (ana, \textbf{ana}), & (ano, \textbf{ano}) \\ (anda, \textbf{anda}), & (ando, \textbf{ando}) \end{array} \right\}$$

$\mathtt{common\_out}_S(a) = \mathtt{lcp}(\{ama, amo, ana, ano, anda, ando\})$

common_out

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the lcp of all $t(wv)$ that are in S.

$$\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\begin{aligned} \texttt{common\_out}_S(a) &= \texttt{lcp}(\{ama, amo, ana, ano, anda, ando\}) \\ &= a \end{aligned}$$

common_out

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the $\texttt{lcp}$ of all $t(wv)$ that are in S.

$$\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$\texttt{common\_out}_S(an) = ???$

common_out

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the lcp of all $t(wv)$ that are in S.

$$\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (\textbf{an}pa, ama), & (\textbf{an}po, amo) \\ (\textbf{an}a, ana), & (\textbf{an}o, ano) \\ (\textbf{an}da, anda), & (\textbf{an}do, ando) \end{array} \right\}$$

$\texttt{common\_out}_S(an) = ???$

common_out

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the lcp of all $t(wv)$ that are in S.

$$\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (\textbf{an}pa, \textbf{ama}), & (\textbf{an}po, \textbf{amo}) \\ (\textbf{an}a, \textbf{ana}), & (\textbf{an}o, \textbf{ano}) \\ (\textbf{an}da, \textbf{anda}), & (\textbf{an}do, \textbf{ando}) \end{array} \right\}$$

$\texttt{common\_out}_S(an) = ???$

`common_out`

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the `lcp` of all $t(wv)$ that are in S.

$$\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (anpa, \textbf{ama}), & (anpo, \textbf{amo}) \\ (ana, \textbf{ana}), & (ano, \textbf{ano}) \\ (anda, \textbf{anda}), & (ando, \textbf{ando}) \end{array} \right\}$$

$$\texttt{common\_out}_S(an) = \texttt{lcp}(\{ama, amo, ana, ano, anda, ando\})$$

common_out

Definition

The common output of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the $\texttt{lcp}$ of all $t(wv)$ that are in S.

$$\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Delta^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\begin{aligned} \texttt{common\_out}_S(an) &= \texttt{lcp}(\{ama, amo, ana, ano, anda, ando\}) \\ &= a \end{aligned}$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$\texttt{min\_change}_S(n, a) =$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \begin{Bmatrix} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{Bmatrix}$$

$\texttt{min\_change}_S(n, a) = \texttt{common\_out}_S(a)^{-1}\texttt{common\_out}_S(an)$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{cc} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\begin{aligned} \texttt{min\_change}_S(n, a) &= \texttt{common\_out}_S(a)^{-1}\texttt{common\_out}_S(an) \\ &= a^{-1}a \end{aligned}$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{ll} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\begin{aligned} \texttt{min\_change}_S(n, a) &= \texttt{common\_out}_S(a)^{-1}\texttt{common\_out}_S(an) \\ &= a^{-1}a \\ &= \lambda \end{aligned}$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\mathtt{min\_change}_S(\sigma, w) = \begin{cases} \mathtt{common\_out}_S(\sigma) & \text{if } w = \lambda \\ \mathtt{common\_out}_S(w)^{-1}\mathtt{common\_out}_S(w\sigma) & \text{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{ll} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$\mathtt{min\_change}_S(p, an) =$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\mathtt{min\_change}_S(\sigma, w) = \begin{cases} \mathtt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \mathtt{common\_out}_S(w)^{-1}\mathtt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{rr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\mathtt{min\_change}_S(p, an) = \mathtt{common\_out}_S(an)^{-1}\mathtt{common\_out}_S(anp)$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (\mathbf{anp}a, \mathbf{am}a), & (\mathbf{anp}o, \mathbf{am}o) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\texttt{min\_change}_S(p, an) = \texttt{common\_out}_S(an)^{-1}\texttt{common\_out}_S(anp)$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \text{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \text{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{ll} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\begin{aligned} \texttt{min\_change}_S(p, an) &= \texttt{common\_out}_S(an)^{-1}\texttt{common\_out}_S(anp) \\ &= a^{-1}am \end{aligned}$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } \ w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{ll} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\begin{aligned} \texttt{min\_change}_S(p, an) &= \texttt{common\_out}_S(an)^{-1}\texttt{common\_out}_S(anp) \\ &= a^{-1}am \\ &= m \end{aligned}$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\mathtt{min\_change}_S(\sigma, w) = \begin{cases} \mathtt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \mathtt{common\_out}_S(w)^{-1}\mathtt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$\mathtt{min\_change}_S(a, an) =$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\texttt{min\_change}_S(a, an) = \texttt{common\_out}_S(an)^{-1}\texttt{common\_out}_S(ana)$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{cc} (anpa, ama), & (anpo, amo) \\ (\mathbf{ana}, \mathbf{ana}), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\texttt{min\_change}_S(a, an) = \texttt{common\_out}_S(an)^{-1}\texttt{common\_out}_S(ana)$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{ll} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\begin{aligned} \texttt{min\_change}_S(a, an) &= \texttt{common\_out}_S(an)^{-1}\texttt{common\_out}_S(ana) \\ &= a^{-1}ana \end{aligned}$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\mathtt{min\_change}_S(\sigma, w) = \begin{cases} \mathtt{common\_out}_S(\sigma) & \text{if } w = \lambda \\ \mathtt{common\_out}_S(w)^{-1}\mathtt{common\_out}_S(w\sigma) & \text{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{ll} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\begin{aligned} \mathtt{min\_change}_S(a, an) &= \mathtt{common\_out}_S(an)^{-1}\mathtt{common\_out}_S(ana) \\ &= a^{-1}ana \\ &= na \end{aligned}$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$\texttt{min\_change}_S(d, an) =$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\texttt{min\_change}_S(d, an) = \texttt{common\_out}_S(an)^{-1}\texttt{common\_out}_S(and)$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{lr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (\mathbf{and}a, \mathbf{and}a), & (\mathbf{and}o, \mathbf{and}o) \end{array} \right\}$$

$$\texttt{min\_change}_S(d, an) = \texttt{common\_out}_S(an)^{-1}\texttt{common\_out}_S(and)$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \text{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \text{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{rr} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\begin{aligned} \texttt{min\_change}_S(d, an) &= \texttt{common\_out}_S(an)^{-1}\texttt{common\_out}_S(and) \\ &= a^{-1}and \end{aligned}$$

min_change

Definition

The minimal change in the output in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is,

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \textit{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \textit{otherwise} \end{cases}$$

Example

Consider the following sample,

$$S = \left\{ \begin{array}{ll} (anpa, ama), & (anpo, amo) \\ (ana, ana), & (ano, ano) \\ (anda, anda), & (ando, ando) \end{array} \right\}$$

$$\begin{aligned} \texttt{min\_change}_S(d, an) &= \texttt{common\_out}_S(an)^{-1}\texttt{common\_out}_S(and) \\ &= a^{-1}and \\ &= nd \end{aligned}$$

Continued by Salam