# Failure Transducers and Applications in Knowledge-Based Text Processing

**Stoyan Mihov**
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
25A, Acad. G. Bonchev Str.,
Sofia 1113, Bulgaria
stoyan@lml.bas.bg

**Klaus U. Schulz**
Centrum für Informations- und Sprachverarbeitung (CIS)
Ludwig-Maximilians-Universität München
Oettingenstr. 67,
80538 München, Germany
schulz@cis.uni-muenchen.de

**Abstract**

Finite-state devices encoding lexica and related knowledge bases often become very large. A well-known technique for reducing the size of finite-state automata is the use of failure transitions. Here we generalize the concept of failure transitions for finite-state automata to the case of subsequential transducers. Failure transitions in the new sense do not have input but may produce output. As an application field for failure transducers we consider text rewriting with large rewrite lexica under the leftmost-longest replacement strategy. It is shown that using failure transducers leads to a huge space reduction compared to the use of standard subsequential transducers. As a concrete example we show how all Wikipedia concepts in an input text can be linked in an online manner with the Wikipedia pages of the concepts using failure transducers.

## 1 Introduction

A wellknown technique for reducing the size of large finite-state automata is the use of failure transitions (Aho and Corasick, 1975; Mohri, 1995; Crochemore and Hancart, 1997; Kourie et al., 2012; Björklund et al., 2014). While automata help to find strings in text, more advanced text processing tasks are often based on knowledge bases that provide information on characteristic portions of input texts (endings, words, phrases, etc.). Using this information, given input texts are translated to a new output form. Examples for this form of "text rewriting" include various forms of tagging, stemming, and (linguistic, semantic,..) annotation (KESA, 2016).

There are a number of efficient techniques for representing a finite dictionary of string entries with their corresponding mappings as finite-state machines and transducers (Mihov and Maurel, 2001; Daciuk et al., 2010). These techniques can produce a very compact representation of the dictionary. But in order to perform *text rewriting* based on the dictionary one has to traverse the dictionary starting from each text position and in addition apply a conflict resolution strategy. Therefore the time complexity for text rewriting is given by the length of the text multiplied by the maximal length of a dictionary entry.

Deterministic finite-state transducers offer an elegant framework to solve such a text processing task in a more efficient way. In (Mihov and Schulz, 2007) we considered "rewriting dictionaries", i.e. collections of strings where for each entry a replacement value (another string) is specified. We showed how to translate a given rewriting dictionary into a subsequential finite-state transducer that may be used to replace all occurrences of dictionary entries in a text by the replacements with only one traversal of the text by the transducer. Using this solution, the time complexity for text rewriting is linear in the length of the text and does not depend on the dictionary. For resolving conflicts between overlapping entries the leftmost-largest rewriting strategy is used. However, when using large rewriting dictionaries the size of the resulting subsequential transducer can become very large. A similar technique is used by Schmitz in (Schmitz, 2011) for constructing subsequential transducers that represent part-of-speech rules.

In this paper we introduce f-transducers, a new kind of deterministic transducer with failure transitions. A failure transition in our sense does not consume input, but it is essential that it may produce output. We show how to translate a given rewriting dictionary into an f-transducer that has the same functionality as the subsequential finite-state transducer obtained in (Mihov and Schulz, 2007). In this way, a huge space reduction is obtained for large rewriting dictionaries. Since transitions in transducers come with output, saving transitions has even a larger benefit than in the automaton case. As a concrete application we consider a rewriting dictionary with 8 million entries where each title of a page of the English Wikipedia obtains a link text with anchor on the corresponding page of the concept. The f-transducer obtained from the translation runs over a text and replaces every mentioning of a Wikipedia concept by a link to the Wikipedia page. In this way, texts can be linked to the Wikipedia in an online-manner.

We start with formal preliminaries in Section 2. In Section 3 we introduce failure transducers. Section 4 presents the construction of f-transducers for text rewriting. The algorithm and complexity analysis of our construction is given in Section 5. Section 6 describes the annotation of concept names with Wikipedia. We finish with a short conclusion in Section 7.

## 2 Formal Preliminaries

An *alphabet* is a finite set $\Sigma$ of symbols. Words of length $n \geq 0$ over an alphabet $\Sigma$ are introduced as usual and written $a_1 \dots a_n$ ($a_i \in \Sigma$). The unique word of length 0 is written $\varepsilon$. As usual, $\Sigma^*$ denotes the set of all words over $\Sigma$. The concatenation of two words $u, v \in \Sigma^*$ is written $u \cdot v$ or $uv$.

**Definition 2.1** A *deterministic finite-state automaton* is a tuple

$$\mathcal{A} = \langle \Sigma, Q, i, F, \delta \rangle$$

where $\Sigma$ is an alphabet, $Q$ is a finite set of states, $i \in Q$ is the start state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \to Q$ is a partial function called the transition function. Let $\mathcal{A} = \langle \Sigma, Q, i, F, \delta \rangle$ be a deterministic finite-state automaton. The *generalized transition function* is the partial function $\delta^* : Q \times \Sigma^* \to Q$ inductively defined as

- $\delta^*(q, \varepsilon) := q$ for all $q \in Q$,
- $\delta^*(q, w\sigma) := \delta(\delta^*(q, w), \sigma)$ for all $q \in Q$, $w \in \Sigma^*$, $\sigma \in \Sigma$ such that $\delta^*(q, w)$ and $\delta(\delta^*(q, w), \sigma)$ are defined.

The *language accepted by* $\mathcal{A}$ is $L(\mathcal{A}) := \{w \in \Sigma^* \mid \delta^*(i, w) \in F\}$.

**Definition 2.2** A *failure automaton* or *f-automaton* is a tuple

$$\mathcal{FA} = \langle \Sigma, Q, i, F, \delta, f \rangle$$

where $\langle \Sigma, Q, i, F, \delta, f \rangle$ is a deterministic finite-state automaton and $f : Q \to Q$ is a partial function called the *failure function*. Let $\mathcal{FA} = \langle \Sigma, Q, q_0, F, \delta, f \rangle$ be an f-automaton. The *completed transition function* $\delta_f : Q \times \Sigma \to Q$ is the least (with respect to $\subseteq$) function $\delta' : Q \times \Sigma \to Q$ such that $\delta'(q, \sigma) :=$

$$\begin{cases} \delta(q, \sigma) & \text{if } \delta(q, \sigma) \text{ is defined,} \\ \delta'(f(q), \sigma) & \text{otherwise, if } f(q) \text{ is defined.} \end{cases}$$

Similarly as $\delta$ and $f$ also $\delta_f$ is a partial function. The *generalized completed transition function* is the (partial) function $\delta_f^* : Q \times \Sigma^* \to Q$ inductively defined as

1. for all $q \in Q$: $\delta_f^*(q, \varepsilon) := q$,
2. for all $q \in Q$, $u \in \Sigma^*$ and $\sigma \in \Sigma$ such that $\delta_f^*(q, u)$ and $\delta_f(\delta_f^*(q, u), \sigma)$ are defined: $\delta_f^*(q, u\sigma) := \delta_f(\delta_f^*(q, u), \sigma)$.

The *language* of the f-automaton $\mathcal{FA}$ is defined as

$$L(\mathcal{FA}) = \{w \in \Sigma^* \mid \delta_f^*(q_0, w) \in F\}.$$

**Definition 2.3** A *subsequential transducer* is a tuple $\mathcal{T} = \langle \Sigma, Q, q_0, F, \delta, \lambda, \Psi \rangle$ where $\langle \Sigma, Q, q_0, F, \delta \rangle$ is a deterministic finite-state automaton, $\lambda : Q \times \Sigma \to \Sigma^*$ is a partial function called the *transition output function* and $\Psi : F \to \Sigma^*$ is a total function called the *state output function*. The domains of $\delta$ and $\lambda$ must coincide. The *generalized transition function* $\delta^*$ is defined as above. The *generalized output function* is the partial function $\lambda^* : Q \times \Sigma^* \to \Sigma^*$ defined as

- $\lambda^*(q, \varepsilon) := \varepsilon$ for all $q \in Q$,
- $\lambda^*(q, w\sigma) := \lambda^*(q, w) \cdot \lambda(\delta^*(q, w), \sigma)$ for all $q \in Q$, $w \in \Sigma^*$, $\sigma \in \Sigma$ such that $\delta^*(q, w)$ and $\delta(\delta^*(q, w), \sigma)$ are defined.

The sets $L_{inp}(\mathcal{T}) := \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$ and $L(\mathcal{T}) :=$

$$\{\langle w, \lambda^*(q_0, w) \cdot \Psi(\delta^*(q_0, w)) \rangle \mid \delta^*(q_0, w) \in F\}$$

are respectively called the *input language* and the *function represented by* $\mathcal{T}$.

The notion of paths in a finite-state device and the length of a path are introduced as usual.

**Definition 2.4** A *position* of $t \in \Sigma^*$ is a pair $\langle u, v\rangle$ such that $t = uv$. An *infix occurrence* (of the infix $v$) in $t \in \Sigma^*$ is a triple $\langle u, v, w\rangle$ such that $t = uvw$. An infix occurrence $\langle u_1, v_1, w_1\rangle$ of the text $t$ *blocks* another infix occurrence $\langle u_2, v_2, w_2\rangle$ of $t$ if $|u_1| < |u_2| < |u_1 v_1|$. In this case we write $\langle u_1, v_1, w_1\rangle <_{ov} \langle u_2, v_2, w_2\rangle$ and say that the two infix occurrences *overlap*. A set $A$ of infix occurrences of the text $t$ is said to be *non-overlapping* if two distinct infix occurrences of $A$ never overlap.

**Definition 2.5** Let $A, B$ be two sets of infix occurrences of the text $t$. We define
$\mathit{AFTER}(A, B) :=$

$$\{\langle u, v, w\rangle \in A \mid \forall \langle u', v', w'\rangle \in B : |u'v'| \leq |u|\}$$

$\mathit{LEFTMOST}(A) :=$

$$\{\langle u, v, w\rangle \in A \mid \forall \langle u', v', w'\rangle \in A : |u| \leq |u'|\}$$

$\mathit{LONGEST}(A) :=$

$$\{\langle u, v, w\rangle \in A \quad \mid \quad \forall \langle u', v', w'\rangle \in A : |u| \neq |u'| \vee |v'| \leq |v|\}$$

**Definition 2.6** Let $A$ be set of infix occurrences of the text $t$. The *subset of leftmost-longest infix occurrences of* $A$ is the set $\mathit{LML}(A) := \bigcup_{i=0}^{\infty} \mathcal{V}_i$ where $\mathcal{V}_0 := \emptyset$ and $\mathcal{V}_{i+1} :=$

$$\mathcal{V}_i \cup \mathit{LONGEST}(\mathit{LEFTMOST}(\mathit{AFTER}(A, \mathcal{V}_i))).$$

## 3 Failure transducers

Failure transducers, or f-transducers, represent a kind of deterministic transducer with a failure transition function. When applying a failure transition during text traversal, an empty part of the input is consumed. However, a non-empty output string may be produced. We start with an illustrating example.

**Example 3.1** Consider the transducer in the upper part of Figure 1. Similarly as State 1, State 2 has outgoing transitions for input letters $a$, $b$, and $c$. Corresponding transitions lead to the same state. After introducing a failure transition from 2 to 1 with output $D$ we can eliminate the standard transitions departing from State 2. The new transducer with failure transitions is shown below. Note that both the number of transitions and the size of the output representation has been reduced.

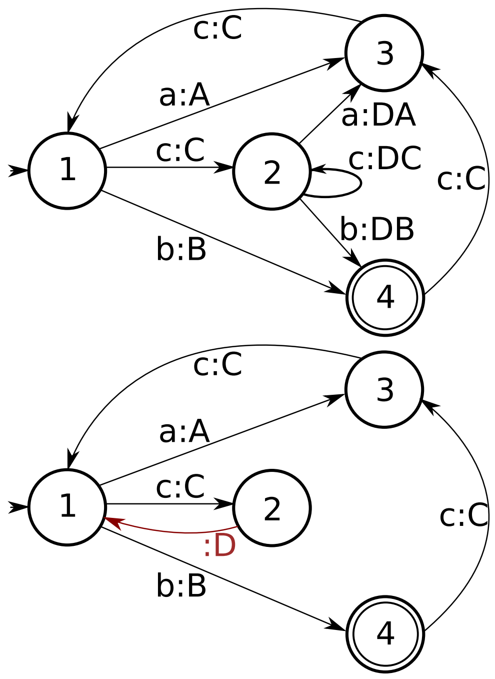

Figure 1: Illustration for Example 3.1: Compressing deterministic finite-state transducers using failure transitions.

When reading, say, symbol $a$ in State 2, we follow the failure link to 1, producing output $D$. Then we use the $a$-transition from 1 to arrive at State 3. The total output is $DA$. The example might appear artificial since with other outputs we could not use the same technique. However, we shall see later that moving parts of the output to the failure transitions is often possible.

Transducers that are used for rewriting texts need to accept arbitrary strings. For this reason we do not introduce a special set of final states in our formalization of failure-transducers. As a matter of fact, generalizations are possible.

**Definition 3.2** An *f-transducer* is a tuple

$$\mathcal{FT} = \langle \Sigma, Q, q_0, \delta, \lambda, \varphi, f\rangle$$

where $\Sigma$ is a finite alphabet, $Q$ is a set of states, $q_0 \in Q$ is the start state, $\delta : Q \times \Sigma \to Q$ is the deterministic transition function, $\lambda : Q \times \Sigma \to \Sigma^*$ is the transition output function, $\varphi : Q \to \Sigma^*$ is the *failure transition output function*, and $f : Q \to Q$ is the *failure transition function*. The following conditions must hold:

1. $\varphi$ and $f$ are partial functions such that $\mathit{dom}(\varphi) = \mathit{dom}(f)$.

2. the domains of $\delta$ and $\lambda$ are identical, i.e., $dom(\delta) = dom(\lambda)$.

**Definition 3.3** Let $\mathcal{FT} = \langle \Sigma, Q, q_0, \delta, \lambda, \varphi, f\rangle$ be an f-transducer. The *completed transition function* $\delta_f : Q \times \Sigma \to Q$ and the *generalized completed transition function* $\delta_f^* : Q \times \Sigma^* \to Q$ are defined as in Definition 2.2. The *completed transition output function* $\lambda_f : Q \times \Sigma \to \Sigma^*$ is defined as the least (with respect to $\subseteq$) function $\lambda' : Q \times \Sigma \to \Sigma^*$ such that $\lambda'(q, \sigma) :=$

$$\begin{cases} \lambda(q,\sigma) & \text{if } \delta(q,\sigma) \text{ is defined,} \\ \varphi(q)\lambda'(f(q),\sigma) & \text{otherwise, if } f(q) \text{ is defined.} \end{cases}$$

The *generalized completed transition output function* $\lambda_f^* : Q \times \Sigma^* \to \Sigma^*$ is inductively defined as

1. For all $q \in Q$: $\lambda_f^*(q, \varepsilon) := \varepsilon$,
2. For all $q \in Q$, $u \in \Sigma^*$ and $\sigma \in \Sigma$: $\lambda_f^*(q, u\sigma) := \lambda_f^*(q,u)\lambda_f(\delta_f^*(q,u),\sigma)$.

The *final state output function* $\Psi_f : Q \to \Sigma^*$ is defined as $\Psi_f(q) :=$

$$\begin{cases} \varepsilon & \text{if } f(q) \text{ is undefined,} \\ \varphi(q)\Psi_f(f(q)) & \text{otherwise.} \end{cases}$$

Let $t$ be a text. The *output* of the f-transducer $\mathcal{FT}$ for input $t$ is

$$O_{\mathcal{FT}}(t) := \lambda_f^*(q_0, t)\Psi_f(\delta_f^*(q_0, t)).$$

Informally, the way how an f-transducer processes a text $t$ can be described as follows: starting from the start state and using the deterministic transition function $\delta$ we read the symbols of the text. The output at each transition is defined by the transition output function. When reaching a state $p$ and a text symbol $\sigma$ such that $\delta(p, \sigma)$ is not defined we apply a series of failure transitions until we arrive at a state $p_n$ such that $\delta(p_n, \sigma) = p'$ is defined. The output produced on this intermediate walk has two parts. The first part is the concatenation of all failure transition outputs of the states $p, \ldots, p_n$ visited. The second part is given by the transition output of the final $\sigma$-transition. Finally, when arriving at state $q$ at the end of the text, we apply a series of failure transitions, producing failure transition outputs, until we reach a state for which the failure function is not defined.

The following lemma shows that an f-transducer $\mathcal{FT}$ in a natural way defines a corresponding subsequential transducer with the same output function. The proof is a direct consequence of our definition of the output function $O_{\mathcal{FT}}$.

**Lemma 3.4** *Let $\mathcal{FT} = \langle \Sigma, Q, q_0, \delta, \lambda, \varphi, f\rangle$ be an f-transducer, let $\delta_f$, $\lambda_f$, and $\Psi_f$ as above. Then $\mathcal{T}_{\mathcal{FT}} := \langle \Sigma, Q, q_0, Q, \delta_f, \lambda_f, \Psi_f\rangle$ is a subsequential transducer and we have $O_{\mathcal{FT}} = O_{\mathcal{T}_{\mathcal{FT}}}$.*

In what follows we consider failure transducers $\mathcal{FT}$ where each state $q$ can be reached from the start state. The depth of a state $q \in Q$, denoted $d(q)$, is the minimal length of a path from start $q_0$ to $q$.

**Definition 3.5** A *backwards f-transducer* is an f-transducer

$$\mathcal{FT} = \langle \Sigma, Q, q_0, \delta, \lambda, \varphi, f\rangle$$

such that for every $q \in Q$ we have $d(q) > d(f(q))$.

**Proposition 3.6** *The time complexity (assuming a random access machine) for rewriting a word $\alpha$ of length $n$ to a word $\beta$ of length $m$ by a backwards f-transducer is $O(n + m)$ and does not depend on the size of the transducer.*

The simple proof is omitted.

## 4 From rewrite dictionaries to f-transducers

As an application field for f-transducers we now look at text rewriting using dictionaries of a particular form.

**Definition 4.1** (Mihov and Schulz, 2007) A *rewrite dictionary* is a pair $\mathcal{D} = (D, \Sigma)$ where $\Sigma$ is an alphabet and $D$ is a finite mapping of words over $\Sigma$. The mapping can be represented in the form

$$D = \{\alpha_i \mapsto \beta_i \mid 1 \leq i \leq k\}$$

such that $\alpha_i \neq \alpha_j$ for $i \neq j \in \{1, 2, \ldots, k\}$. Each string $\alpha_i$ is called an *entry* or an *original* of $\mathcal{D}$, and $\beta_i$ is called the *replacement value* for $\alpha_i$ ($i = 1, \ldots, k$).

**Definition 4.2** Let $t \in \Sigma^*$ be a text and $\mathcal{D} = (D, \Sigma)$ be a rewrite dictionary. A *rewrite occurrence* of $\mathcal{D}$ in $t$ is an infix occurrence $\langle u, v, w\rangle_t$ such that $v \in dom(D)$. By $\mathcal{C}_t^{\mathcal{D}}$ we denote the set of all rewrite occurrences of $D$ in the text $t$. The *global rewriting function associated with $D$* is the mapping $L(\mathcal{D}) : \Sigma^* \to \Sigma^*$ that given an input text $t$ replaces each leftmost-longest infix occurrence of $\mathcal{C}_t^{\mathcal{D}}$ in $t$ by the corresponding replacement value.

**Example 4.3** (From (Mihov and Schulz, 2007)). Let $\mathcal{D}$ denote the rewriting dictionary with alphabet $\Sigma := \{a, b, c, 1, 2, 3, 4, 5\}$ and mapping $D$ of the form

$$\begin{array}{rcl} (a & \mapsto & 1) \\ (ab & \mapsto & 2) \\ (abcc & \mapsto & 3) \\ (babc & \mapsto & 4) \\ (c & \mapsto & 5) \end{array}$$

The leftmost-longest infix occurrences of $\mathcal{C}_t^{\mathcal{D}}$ in the text $t = abcbbbabccb$ are

$$\begin{array}{c} \langle \epsilon, ab, cbbbabccb \rangle \\ \langle ab, c, bbbabccb \rangle \\ \langle abcbb, babc, cb \rangle \\ \langle abcbbbabc, c, b \rangle \end{array}$$

and $L(\mathcal{D})(t) = 25bb45b$.

We now describe a procedure for translating a rewrite lexicon $\mathcal{D}$ into a backwards f-transducer $\mathcal{FT}$ such that the global rewriting function $L(\mathcal{D})$ associated with $\mathcal{D}$ and $O_{\mathcal{FT}}$ are identical. The construction is a variant of the construction presented (Mihov and Schulz, 2007) for translating rewrite dictionaries into standard subsequential transducers. As in (Mihov and Schulz, 2007) we proceed in two steps.

**Step 1.** Given the rewrite lexicon $\mathcal{D}$, as in (Mihov and Schulz, 2007) we build a trie transducer $\mathcal{T}_D$ representing the domain of the lexicon mapping $D$. The final states of $\mathcal{T}_D$ correspond to the entries ("originals") of $\mathcal{D}$, and the failure transition output of each final state is defined as the image of the entry. The transition output for each transition is the empty string $\varepsilon$. The trie transducer thus represents the finite mapping $D$ given by $\mathcal{D}$.

**Step 2.** The second step, where we build the failure transducer $\mathcal{FT}$ representing the global rewriting function for $D$, is based on a procedure where we visit the states of the trie transducer in a breadth-first manner, starting at the initial state $q_0$. We first complete the initial state $q_0$ adding loop transitions with any symbol $\sigma \in \Sigma$ such that there is no outgoing $\sigma$-transition from $q_0$ in the trie. The transition output for a loop transitions with symbol $\sigma$ is $\sigma$. For all states $q$ which are direct ancestors of $q_0$ i.e. such that $q = \delta(q_0, \sigma)$ we define $f(q) := q_0$ and $\varphi(q) := \sigma$ if $q$ is not final. In case $q$ is final the function $\varphi(q)$ is already defined.

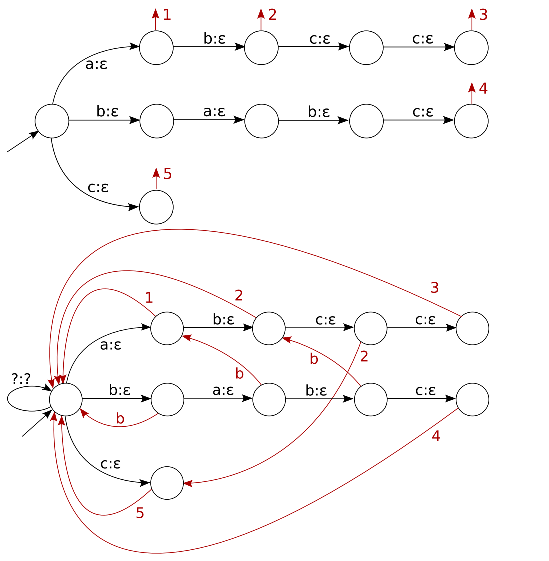

Figure 2: Trie-transducer (Step 1) and f-transducer (Step 2) obtained for the rewrite lexicon from Example 4.4.

Assume now that for the state $q \neq q_0$ we have already defined $f(q) = p'_q$ and $\varphi(q) = \gamma'_q$. Let $q' = \delta(q, \sigma)$ be an ancestor of $q$ in the trie.

*Case a.* If $q'$ is a final state of the trie transducer - i.e., if the failure transition output $\varphi(q')$ for $q$ is already defined - we just define $f(q') := q_0$.

*Case b.* In the other case we define $f(q') := \delta_f(p'_q, \sigma)$ and $\varphi(q') := \gamma'_q \cdot \lambda_f(p'_q, \sigma)$. The definition of $\delta_f$ shows that we find state $f(q')$ by starting from $f(q) = p'_q$ and applying failure transitions until we arrive at a state $p_n$ such that $\delta(p_n, \sigma) = f(q')$ is defined.

**Example 4.4** As an illustration for the translation of rewrite lexica into f-transducer we use the rewite lexicon from Example 4.3. The resulting trie transducer (Step 1) and f-transducer (Step 2) are shown in Figure 2. When processing text $t = abcbbbabccb$ we first read prefix $abc$ with no output. Then two failure transitions produce output 25 before we can read the next letter $b$ from the start. After reading $t$ we have produced output $25bb45$ and the current state is the $b$-successor of the start. We have to add the final state output for this state, which is given by the failure transition output $b$ (cf. Def. 3.3). The total output is $25bb45b$ as in Example 4.3.

**Remark 4.5** For the following correctness proof we sketch Step 2 of the parallel construction of a subsequential transducer $\mathcal{T}$ in (Mihov and Schulz, 2007). Recall that in this case for each state $q$ and each symbol $\sigma \in \Sigma$ such that $q$ does not have a $\sigma$ transition in the transducer trie $\mathcal{T}_D$ a new $\sigma$-transition with suitable output needs to be added. For $q_0$ the procedure is as above (as for each state, also $q_0$ is made final). Consider a state $q_0 \neq q$ processed during Step 2. Let *plab*$(q) = a_1 \ldots a_r$ denote the label of the path $\pi$ in the trie from $q_0$ to $q$. The *skip part of plab*$(q) = a_1 \ldots a_r$, denoted $u_1$, is:

1. $a_1$ if the state sequence $\pi$ does not contain any final state of $\mathcal{T}_D$, and
2. $a_1 \cdots a_f$ ($f \leq r$) if this prefix of *plab*$(q)$ leads from $q_0$ to the last final state of $\pi$.

The *read part of plab*$(q)$ (denoted $u_2$) is the remaining part $u_2$ of *plab*$(q) = u_1 u_2$. Note that the read part is the empty word if $q$ is a final state of $\mathcal{T}_D$ or if $q$ is a direct successor of the start state $q_0$. The *failure state for* $q$ is the state $p'$ obtained when traversing the transducer $\mathcal{T}$ with the read part $u_2$, starting from $q_0$. The construction order guarantees that $u_2$ can be completely read in the preliminary version of the subsequential transducer $\mathcal{T}$ computed up to this point since the length of the read part $u_2$ is smaller than the length of *plab*$(q)$. The *output prefix for* $q$ is the string $\gamma'$ that represents the concatenation of (i) the output of the transducer $\mathcal{T}$ for the skip part $u_1$ (either $a_1$ or the substitute for the lexical entry $a_1 \cdots a_f$, cf. cases above) and (ii) the transition output of the transducer $\mathcal{T}$ for the read part $u_2$ when starting from $q_0$. With these notions, the processing of $q$ can be described in the following way:

1. The state output for $q$ is the concatenation of the output prefix $\gamma'$ with the state output of the failure state $p'$.
2. If a new $\sigma$-transition from $q$ is needed, the target state is the $\sigma$-successor of the failure state for $q$ (it always exists since the failure state has smaller depth than $q$). The transition output for the new $\sigma$-transition from $q$ is the concatenation of the output prefix $\gamma'$ with the transition output of the transition with label $\sigma$ from the failure state $p'$.

**Correctness proof.** Because of space limitations, an independent and fully selfcontained proof cannot be given here. However, using the correctness of the parallel construction in (Mihov and Schulz, 2007) (shown in the paper) and Remark 4.5 we can prove correctness of the new construction. Let

$$\mathcal{FT} = \langle \Sigma, Q, q_0, \delta, \lambda, \varphi, f \rangle$$

denote the f-transducer obtained. Let

$$\mathcal{T} = \langle \Sigma, Q, q_0, Q, \delta_T, \lambda_T, \Psi_T \rangle$$

denote the subsequential transducer obtained from the construction described in (Mihov and Schulz, 2007). The following lemma captures some parallelisms between the two devices. We use the notation introduced in Definition 3.3.

**Lemma 4.6** *1. For any state $q \neq q_0$ the state $f(q)$ is the failure state of $q$ in the sense of Remark 4.5. We have $d(f(q)) < d(q)$.*

2. *For any state $q$ and any letter $\sigma$ we have $\delta_T(q, \sigma) = \delta_f(q, \sigma)$. Furthermore $d(\delta_T(q, \sigma)) \leq d(q) + 1$.*
3. *For any state $q \neq q_0$ the failure transition output $\varphi(q)$ is the output prefix $\gamma'(q)$ in the sense of Remark 4.5.*
4. *For any state $q$ and any letter $\sigma$ we have $\lambda_T(q, \sigma) = \lambda_F(q, \sigma)$.*

The proof for Lemma 4.6 is given in below. Looking at Part 3 of Lemma 4.6 it is simple to see that for each state $q$ we have $\Psi_T(q) = \Psi_f(q)$. When we now compare the outputs produced for a text $t$ by $\mathcal{T}$ and $\mathcal{FT}$ respectively we find, using Lemma 4.6, that

$$\begin{aligned} O_{\mathcal{T}}(t) &= \lambda_T^*(q_0, t) \cdot \Psi_T(\delta_T(q_0, t)) \\ &= \lambda_f^*(q_0, t) \cdot \Psi_f(\delta_f(q_0, t)) = O_{\mathcal{FT}}(t). \end{aligned}$$

In (Mihov and Schulz, 2007) it has been shown that $O_{\mathcal{T}}$ represents the global rewrite function $L(\mathcal{D})$ for the rewrite lexicon $\mathcal{D}$ under the leftmost largest strategy in the sense of Definition 4.2. Hence the same holds for $O_{\mathcal{FT}}$, which shows that the new construction is correct. $\square$

**Proof of Lemma 4.6.** Recall that for a state $q$, the length of the unique path from $q_0$ to $q$ in the transducer trie is denoted $d(q)$. The proof is by induction on $d(q)$. If $d(q) = 0$ we have $q = q_0$. In this case, Claims 2 and 4 are obvious since we have $\delta_T(q_0, \sigma) = \delta_f(q_0, \sigma) =$

$$\begin{cases} \delta(q_0, \sigma) & \text{if } \delta(q_0, \sigma) \text{ is defined,} \\ q_0 & \text{otherwise.} \end{cases}$$

and $\lambda_T(q_0,\sigma) = \lambda_F(q_0,\sigma) =$

$$\begin{cases} \varepsilon & \text{if } \delta(q_0,\sigma) \text{ is defined,} \\ \sigma & \text{otherwise.} \end{cases}$$

There is nothing to show as to Claims 1 and 3.

For the induction step consider a successor state $q = \delta(q',\sigma')$. Let $p'$ denote the failure state of $q'$ in the sense of Remark 4.5. Let $p := \delta_T(p',\sigma')$. By induction hypothesis we have (i) $p' = f(q')$ and (ii) $\delta_T(p',\sigma') = \delta_f(p',\sigma')$. Furthermore $d(p') < d(q') < d(q)$ and $d(p) \leq d(p') + 1 < d(q)$.

We show Claim 1. If $q$ is final, then $q_0 = f(q)$ is the failure state of $q$ in the sense of Remark 4.5. In the other case the failure state of $q$ in the sense of Remark 4.5 is $p = \delta_T(p',\sigma') = \delta_f(p',\sigma') = \delta_f(f(q'),\sigma') = f(q)$.

We show Claim 2. Let $\sigma \in \Sigma$. If $\delta(q,\sigma)$ is defined we have $\delta_T(q,\sigma) = \delta(q,\sigma) = \delta_f(q,\sigma)$. In the other case, first consider the case where $q$ is final. Then $q_0$ is the failure state for $q$ and $\delta_T(q,\sigma) = \delta_T(q_0,\sigma) = \delta_f(q_0,\sigma) = \delta_f(f(q),\sigma) = \delta_f(q,\sigma)$. By induction hypothesis $d(\delta_f(q_0,\sigma)) \leq 1$ and thus $d(\delta_T(q,\sigma)) \leq 1 < d(q)+1$. If $q$ is not final, then $p = \delta_T(p',\sigma')$ is the failure state for $q$. We have seen that $d(p) < d(q)$. By induction hypothesis $\delta_T(p,\sigma) = \delta_f(p,\sigma)$. Claim 1 shows that $p = f(q)$. Hence

$$\begin{aligned} \delta_T(q,\sigma) &= \delta_T(p,\sigma) = \delta_f(p,\sigma) \\ &= \delta_f(f(q),\sigma) = \delta_f(q,\sigma). \end{aligned}$$

We have $d(\delta_T(q,\sigma)) = d(\delta_T(p,\sigma)) \leq d(p)+1 \leq d(q)+1$.

We show Claim 3. If $q$ is final, then $\gamma'(q) = \Psi_T(q) = \varphi(q)$ (note that $\Psi_T(q)$ is the state output for $q$ in the transducer trie). In the other case, by induction hypothesis we have $\gamma'(q') = \varphi(q')$ and $\lambda_T(p',\sigma') = \lambda_F(p',\sigma')$. In this situation

$$\begin{aligned} \gamma'(q) &= \gamma'(q') \cdot \lambda_T(p',\sigma') = \varphi(q') \cdot \lambda_T(p',\sigma') \\ &= \varphi(q') \cdot \lambda_F(p',\sigma') = \varphi(q). \end{aligned}$$

We show Claim 4. Let $\sigma \in \Sigma$. If $\delta(q,\sigma)$ is defined we have $\lambda_T(q,\sigma) = \varepsilon = \lambda_F(q,\sigma)$. In the other case we have $\lambda_T(q,\sigma) = \gamma'(q) \cdot \lambda_T(p,\sigma)$ where $\gamma'(q)$ is the output prefix for $q$ in the sense of Remark 4.5. Using Claim 3 and the induction hypothesis we see that

$$\begin{aligned} \lambda_T(q,\sigma) &= \varphi(q) \cdot \lambda_T(p,\sigma) \\ &= \varphi(q) \cdot \lambda_f(p,\sigma) = \lambda_f(q,\sigma). \Box \end{aligned}$$

## 5 Implementation and complexity analysis

Algorithm 1 presents the pseudo-code of the construction. Clearly, the number of states of the f-transducer is bounded by $||D_I|| + 1$ and the number of transitions is bounded by $2||D_I||$ and does not depend on the alphabet size, where $||D_I|| = \sum_{\langle\alpha,\beta\rangle\in D} |\alpha|$. The complexity of the trie construction is $O(||D||)$, where $||D|| = \sum_{\langle\alpha,\beta\rangle\in D} |\alpha| + |\beta|$. If each output string is represented in the standard way as a sequence of symbols, the space complexity of $\varphi$ can get cubic in $||D||$. Using the technique for tree-based output string representation introduced in (Mihov and Schulz, 2007) the space complexity remains linear in $||D||$. In that case the space and time complexity of the proposed algorithm[1] is $O(||D||)$. In contrast, the space and time complexity of the construction presented in (Mihov and Schulz, 2007) is $O(||D|| \cdot |\Sigma|)$ for an alphabet $\Sigma$. The complexity for rewriting a text $t$ to $t'$ is $O(|t| + |t'|)$.

## 6 Application for Online Hyperlinking Using Link Databases

Document repositories often come with a large number of internal or external links that lead from concepts (entities, references, etc.) mentioned in the text to other web pages. A well-known example is the Wikipedia[2]. On a Wikipedia page, all concepts of the text that are described in more detail in some other article of the Wikipedia are highlighted. When clicking at the concept the visitor is led to the relevant page of the Wikipedia, using an internal link (wikilink). In the same way, arbitrary texts can be linked with the Wikidedia.

To create a wikilink, concept names such as “United Kingdom” or “Kingdom University” mentioned in the text are replaced by anchor elements of the form

```
<a href="/wiki/United_Kingdom">
                    United Kingdom</a>
<a href="/wiki/Kingdom_University">
                    Kingdom University</a>
```

In order to automatize this form of hyperlinking, rewriting dictionaries may be used that list relevant concept names (e.g. “United Kingdom”) together with the corresponding anchor elements

[1]Assuming that in the function `f-transducer` the loop transitions of $q_0$ in Lines 5-6 are not explicitly generated and the loops starting at Line 2 and Line 16 iterate only through existing transitions.

[2]http://en.wikipedia.org/wiki/Main_Page

**Algorithm 1** Construction of an f-transducer for a given alphabet $\Sigma$ and rewrite dictionary $D$.

```
Trie($\Sigma, D$)
  $q_0 \leftarrow new\_state()$
  $Q \leftarrow \{q_0\}$
  $\delta \leftarrow \emptyset$
  $\lambda \leftarrow \emptyset$
  $\varphi \leftarrow \emptyset$
  for $\langle \alpha, \beta \rangle \in D$ do
    $q \leftarrow q_0$
    for each symbol $\sigma$ of $\alpha$ do
      $p \leftarrow \delta(q, \sigma)$
      if $p = nil$ then
        $p \leftarrow new\_state()$
        $Q \leftarrow Q \cup \{p\}$
        $\delta(q, \sigma) \leftarrow p$
        $\lambda(q, \sigma) \leftarrow \varepsilon$
      $q \leftarrow p$
    $F \leftarrow F \cup \{q\}$
    $\varphi(q) \leftarrow \beta$
  return $\langle \Sigma, Q, q_0, \delta, \lambda, \varphi, \emptyset \rangle$
```

```
f-transducer($\Sigma, D$)
  $\langle \Sigma, Q, q_0, \delta, \lambda, \varphi, f \rangle \leftarrow Trie(\Sigma, D)$
  for $\sigma \in \Sigma$ do
    $q \leftarrow \delta(q_0, \sigma)$
    if $q = nil$ then
      $\delta(q_0, \sigma) \leftarrow q_0$
      $\lambda(q_0, \sigma) \leftarrow \sigma$
    else
      if $!\varphi(q)$ then
        $push\_queue(\langle q, q_0, \varphi(q) \rangle)$
      else
        $push\_queue(\langle q, q_0, \sigma \rangle)$
  while not $empty\_queue()$ do
    $\langle q, p'_q, \gamma'_q \rangle \leftarrow pull\_queue()$
    $f(q) \leftarrow p'_q$
    $\varphi(q) \leftarrow \gamma'_q$
    for $\sigma \in \Sigma$ do
      $q' \leftarrow \delta(q, \sigma)$
      if $q' \neq nil$ then
        if $!\varphi(q')$ then
          $push\_queue(\langle q', q_0, \varphi(q') \rangle)$
        else
          $p'' \leftarrow p'_q$
          $\gamma'' \leftarrow \gamma'_q$
          while $\delta(p'', \sigma) = nil$ do
            $\gamma'' \leftarrow concat(\gamma'', \varphi(p''))$
            $p'' \leftarrow f(p'')$
          $push\_queue(\langle q', \delta(p'', \sigma), concat(\gamma'', \lambda(p'', \sigma)) \rangle)$
  return $\langle \Sigma, Q, q_0, \delta, \lambda, \varphi, f \rangle$
```

as a replacement. When processing a text with the rewriting dictionary, each occurrence of a concept name is replaced by the corresponding anchor element as a form of annotation. We note that the rewriting dictionaries used for automated hyperlinking are often extremely large.

Algorithm 1 has been applied for constructing a rewrite f-transducer for annotating texts with the anchor elements of practically all Wikipedia concept names in English[3]. The rewrite dictionary has $8,083,029$ entries and occupies $831$ MB. The corresponding f-transducer was constructed by a not optimized implementation of Algorithm 1 using 64 bits for pointer, number and character representation. The resulting f-transducer has $69,037,940$ states and occupies $11.5$ GB. On an Intel Xeon CPU at $2.40$ GHz the construction time is 63 minutes. Using the constructed f-transducer, 100 MB of text are rewritten in approximately 19 seconds. This makes it possible to maintain huge knowledge bases in memory and rewrite texts in an online matter. In the same way, texts could be linked with any other collection of linked open data.

In order to compare the new construction with the construction presented in (Mihov and Schulz, 2007) we used a 5 MB English correction dictionary with $220,231$ entries. All words in the correction dictionary are over the 26 lower case English letters. The rewrite f-transducer is constructed in 12 seconds and occupies 79 MB. The corresponding subsequential transducer is constructed in 283 seconds and occupies 1170 MB (cf. Table 1).

| | time | size |
|---|---|---|
| subseq. transducer | 283 s | 1170 MB |
| f-transducer | 12 s | 79 MB |

Table 1: Translation of a 5 MB rewrite dictionary into a text rewriting device. Construction times and sizes when using a subsequential transducer and an f-transducer.

## 7 Conclusion

In this paper we introduced the concept of failure transducers. We presented in detail the construction of f-transducers for dictionary based text rewriting under the left-most-longest match strategy for conflict resolution. The advantage of the new construction compared to the method presented in (Mihov and Schulz, 2007) is space and construction time economy. The new construction

[3]There can be multiple concept names for one Wikipedia page.

avoids the increase in complexity which is caused by the enormous number of transitions needed for subsequential transducers when using rewrite dictionary over large alphabets. In our construction the number of ordinary transitions and the number of failure transitions of the f-transducer are bounded by the sum of the lengths of input words in the dictionary and do not depend on the alphabet size. This made it possible to construct an f-transducer for annotating all concept names of the English Wikipedia in a text in online manner on a personal computer. The same technique can be applied to similar forms of knowledge-based text rewriting. In this way, interesting items in input texts can be linked "on demand" in a user-driven online manner to distict targets such as lexica, authority pages, product catalogues and other resources.

## References

Alfred Aho and Margaret Corasick. 1975. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18:333–340.

Henrik Björklund, Johanna Björklund, and Niklas Zechner. 2014. Compression of finite-state automata through failure transitions. *Theoretical Computer Science*, 557:87–100.

Maxime Crochemore and Christophe Hancart. 1997. Automata for matching patterns. In *Handbook of formal languages*, pages 399–462. Springer.

Jan Daciuk, Jakub Piskorski, and Strahil Ristov, 2010. *Scientific Applications of Language Methods*, chapter Natural Language Dictionaries Implemented as Finite Automata, pages 133–204. Imperial College Press.

KESA. 2016. Third international workshop on knowledge extraction and semantic annotation. In *Proceedings ALLDATA 2016 The Second International Conference on Big Data, Small Data, Linked Data and Open Data*, Lisbon, Portugal.

Derrick G. Kourie, Bruce W. Watson, Loek Cleophas, and Fritz Venter. 2012. Failure deterministic finite automata. In *Proceedings of Prague Stringology Conference*, pages 28–41.

Stoyan Mihov and Denis Maurel. 2001. Direct construction of minimal acyclic subsequential transducers. In *Proceedings of the Conference on Implementation and Application of Automata CIAA'2000*, number 2088 in LNCS, pages 217–229. Springer.

Stoyan Mihov and Klaus U. Schulz. 2007. Efficient dictionary-based text rewriting using subsequential transducers. *Natural Language Engineering*, 13(4):353–381, December.

Mehryar Mohri. 1995. Matching patterns of an automaton. In *Combinatorial Pattern Matching*, pages 286–297. Springer.

Sylvain Schmitz. 2011. A note on sequential rule-based pos tagging. In *Proceedings of the 9th International Workshop on Finite State Methods and Natural Language Processing*, FSMNLP '11, pages 83–87, Stroudsburg, PA, USA. Association for Computational Linguistics.