Direct Construction of Minimal Acyclic Subsequential Transducers

Stoyan Mihov[1] and Denis Maurel[2]

[1] Linguistic Modelling Laboratory
LPDP – Bulgarian Academy of Sciences
`stoyan@lml.bas.bg`
[2] LI (Computer Laboratory) – University of Tours
E3i, 64 avenue Jean-Portalis, F37200 Tours, France
`maurel@univ-tours.fr`

Abstract. This paper presents an algorithm for direct building of minimal acyclic subsequential transducer, which represents a finite relation given as a sorted list of words with their outputs. The algorithm constructs the minimal transducer directly – without constructing intermediate tree-like or pseudo-minimal transducers. In NLP applications our algorithm provides significantly better efficiency than the other algorithms building minimal transducer for large-scale natural language dictionaries. Some experimental comparisons are presented at the end of the paper.

1 Introduction

For the application of large-scale dictionaries two major problems have to be solved: fast lookup speed and compact representation. Using automata we can achieve fast lookup by determinization and compact representation by minimization. For providing information for the recognized words we have to construct automata with outputs or transducers. The use of automata with labels on the final states for representation of dictionaries is presented by Dominique Revuz in [8]. In [6, 7] Mehryar Mohri reviews the application of transducers for Natural Language Processing. He compares the benefits using subsequential transducers. The transducers are more compact in some cases and can be combined by composition or other relational operators. The transducers can be applied also for the reverse direction – to find the input words which are mapped to a given output.

In this paper we focus on building the minimal subsequential transducer for a given input list of words with their outputs. This is the procedure required for the initial construction of the transducer representing a dictionary. Earlier presented methods are building temporary transducers for the input list first, and later they have to be minimized. This temporary transducers can be huge compared to the resulting minimized one. For example in [6] Mehryar Mohri writes:

"But, as with automata, one cannot construct directly the p-subsequential transducer representing a large-scale dictionary. The tree construction mentioned above leads indeed to a blow up for a large number of entries. So, here again, one needs first to split the dictionary into several parts, construct the corresponding p-subsequential transducers, minimize them, and then perform the union of these transducers and reminimize the resulting one."

In [3] Denis Maurel is building efficiently the pseudo-minimal subsequential transducer, which can be significantly smaller than the tree-like transducer. The pseudo-minimal transducer has to be additionally minimized. Our experiments are showing that for large-scale dictionaries the pseudo-minimal subsequential transducer is about 10 times larger than the minimized transducer.

In this paper we present an algorithm for building minimal subsequential transducer for a given sorted list without the necessity of building any intermediate non-minimal transducers. The algorithm is a combination of the algorithm for direct construction of minimal acyclic Finite-State automaton given in [2,4] with the methods for construction of minimal subsequential transducers given in [3,5]. The resulting subsequential transducer is minimal.

In comparison with the approach of Mehryar Mohri we don't build minimal intermediate transducers for parts of the dictionary which after deterministic union have to be minimized again. We are proceeding incrementally word by word building the minimal except for the last word transducer.

2 Mathematical Concepts

In this section we present shortly the mathematical basics used in the algorithm for direct construction of minimal subsequential transducers. A more detailed presentation with the corresponding proofs for the minimal except for a word automata is given in [4].

2.1 Subsequential Transducers

Definition 1. *A p-subsequential transducer is a tuple* $\mathcal{T} = \langle \Sigma, \Delta, S, s, F, \mu, \lambda, \Psi \rangle$, *where:*

- Σ *is a finite input alphabet;*
- Δ *is a finite output alphabet;*
- S *is a finite set of states;*
- $s \in S$ *is the starting state;*
- $F \subseteq S$ *is the set of final states;*
- $\mu : S \times \Sigma \rightarrow S$ *is a partial function called the transition function;*
- $\lambda : S \times \Sigma \rightarrow \Delta^*$ *is a partial function called the output function;*
- $\Psi : F \rightarrow 2^{\Delta^*}$ *is the final function. We will require that* $\forall r \in F\ (|\Psi(r)| \leq p)$.

The function μ is extended naturally over $S \times \Sigma^*$ as in the case for finite state automata:

$$\forall r \in S\ (\mu^*(r,\varepsilon) = r)\ ;\ \forall r \in S\ \forall \sigma \in \Sigma^*\ \forall a \in \Sigma\ (\mu^*(r,\sigma a) = \mu(\mu^*(r,\sigma),a)).$$

The function λ is extended over $S \times \Sigma^*$ by the following definition:

$$\forall r \in S\ (\lambda^*(r,\varepsilon) = \varepsilon);\ \forall r \in S\ \forall \sigma \in \Sigma^*\ \forall a \in \Sigma\ (\lambda^*(r,\sigma a) = \lambda^*(r,\sigma)\lambda(\mu^*(r,\sigma),a)).$$

The set $L(\mathcal{T}) = \{\sigma \in \Sigma^* \mid \mu^*(s,\sigma) \in F\}$ is called the input language of the transducer $\mathcal{T}$. The subsequential transducer maps each word from the input language to a set of at most p output words. The output function $O_{\mathcal{T}} : L(\mathcal{T}) \to 2^{\Delta^*}$ of the transducer is defined as follows:

$$\forall \sigma \in L(\mathcal{T})\ (O_{\mathcal{T}}(\sigma) = \lambda^*(s,\sigma) \cdot \Psi(\mu^*(s,\sigma))).$$

Two transducers $\mathcal{T}$ and $\mathcal{T}'$ are called equivalent when $L(\mathcal{T}) = L(\mathcal{T}')$ and $O_{\mathcal{T}} = O_{\mathcal{T}'}$.

Definition 2. *Let $\mathcal{T} = \langle \Sigma, \Delta, S, s, F, \mu, \lambda, \Psi \rangle$ be a subsequential transducer.*

1. *The state $r \in S$ is called reachable from $t \in S$, when $\exists \sigma \in \Sigma^*\ (\mu^*(t,\sigma) = r)$.*
2. *We define the subtransducer starting in $s' \in S$ as:*
   $\mathcal{T}|_{s'} = \langle \Sigma, \Delta, S', s', F \cap S', \mu|_{S' \times \Sigma}, \lambda|_{S' \times \Sigma}, \Psi|_{F \cap S'} \rangle$*, where:*
   $S' = \{r \in S \mid r$ *is reachable from* $s'\}$.
3. *Two states $s_1, s_2 \in S$ are called equivalent, when $\mathcal{A}|_{s_1}$ and $\mathcal{A}|_{s_2}$ are equivalent (when $L(\mathcal{T}|_{s_1}) = L(\mathcal{T}|_{s_2})$ and $O_{\mathcal{T}|_{s_1}} = O_{\mathcal{T}|_{s_2}}$).*

We cannot use directly the minimization algorithms developed for automata because in some cases by moving the output labels (or parts of them) along the paths we can get a smaller transducer. To avoid this we have to use transducers which has the property that the output is pushed back toward the initial state as far as possible. Mehryar Mohri [5] shows that there is a minimal transducer which satisfies this property. We will define this more formally bellow.

With $u \wedge v$ we denote the longest common prefix of the words u and v from Σ^* and with $u^{-1}(uv)$ we denote the word v – the quotient of the left division of uv by u. For the set of words $A = \{a_1, a_2, \ldots, a_n\}$ with $\bigwedge A$ we will denote the word $\bigwedge A = a_1 \wedge a_2 \wedge \ldots \wedge a_n$.

With $D(\mathcal{T})$ we will denote the set of the prefixes of $L(\mathcal{T})$:

$$D(\mathcal{T}) = \{u \in \Sigma^* \mid \exists w \in \Sigma^*\ (uw \in L(\mathcal{T}))\}.$$

We define the function $g_{\mathcal{T}} : D(\mathcal{T}) \to \Delta^*$ as follows:

$$g_{\mathcal{T}}(\varepsilon) = \varepsilon\ \ ;\ \ g_{\mathcal{T}}(u) = \bigwedge_{w \in \Sigma^*\ \&\ uw \in L(\mathcal{T})} \bigwedge O_{\mathcal{T}}(uw), \text{ for } u \in D(\mathcal{T}), u \neq \varepsilon.$$

Now we are ready to define the canonical subsequential transducer.

Definition 3. *The subsequential transducer $\mathcal{T} = \langle \Sigma, \Delta, S, s, F, \mu, \lambda, \Psi \rangle$ is called canonical if the following condition holds:*

$$\forall r \in S\ \forall a \in \Sigma\ \forall \sigma \in \Sigma^*\ ((\mu^*(s,\sigma) = r\ \&\ !\mu(r,a)) \rightarrow \lambda(r,a) = [g_{\mathcal{T}}(\sigma)]^{-1} g_{\mathcal{T}}(\sigma a))$$

We can see that the condition above corresponds to the property that the output is pushed back to the initial state as much as possible.

The subsequential transducer $\mathcal{T}$ is called minimal if any other transducer equivalent to $\mathcal{T}$ has more or equally many states as $\mathcal{T}$.

Theorem 1. *For every subsequential transducer $\mathcal{T}$ there exists a minimal canonical subsequential transducer equivalent to $\mathcal{T}$.*

Theorem 2. *If there are no different equivalent states in the canonical subsequential transducer $\mathcal{T}$ then $\mathcal{T}$ is minimal.*

A more complete presentation of the minimal subsequential transducers can be find in [5]).

2.2 Minimal Except for a Word Subsequential Transducer

Definition 4. *Let $\mathcal{T} = \langle \Sigma, \Delta, S, s, F, \mu, \lambda, \Psi \rangle$ be a subsequential transducer with input language $L(\mathcal{T})$. Then the transducer $\mathcal{T}$ is called* minimal except for the word *$\omega \in \Sigma^*$, when the following conditions hold:*

1. *Every state is reachable from the starting state and from every state a final state is reachable;*
2. *ω is a prefix of the last word in the lexicographical order of $L(\mathcal{T})$;*
   In that case we can introduce the following notations:

$$\omega = w_1^{\mathcal{T}} w_2^{\mathcal{T}} \dots w_k^{\mathcal{T}}\ ,\ \textit{where}\ w_i^{\mathcal{T}} \in \Sigma\ ,\ \textit{for}\ i = 1,2,\dots,k \quad (1)$$

$$t_0^{\mathcal{T}} = s\ ;\ t_1^{\mathcal{T}} = \mu(t_0^{\mathcal{T}}, w_1^{\mathcal{T}})\ ;\ t_2^{\mathcal{T}} = \mu(t_1^{\mathcal{T}}, w_2^{\mathcal{T}})\ ;\ \dots\ ;\ t_k^{\mathcal{T}} = \mu(t_{k-1}^{\mathcal{T}}, w_k^{\mathcal{T}}) \quad (2)$$

$$T = \{t_0^{\mathcal{T}}, t_1^{\mathcal{T}}, \dots, t_k^{\mathcal{T}}\} \quad (3)$$

3. *In the set $S \setminus T$ there are no different equivalent states;*
4. *$\forall r \in S\ \forall i \in \{1,2,\dots,k\}\ \forall a \in \Sigma\ (\mu(r,a) = t_i \leftrightarrow (i > 0\ \&\ r = t_{i-1}\ \&\ a = w_i^{\mathcal{T}}))$;*
5. *$\mathcal{T}$ is a canonical subsequential transducer.*

Example 1. An acyclic 2-subsequential transducer over the input alphabet $\{a,b,c\}$ is given on Figure 1. The input language of the transducer is [{apr, aug, dec, feb, jan, jul}.] The output function of the transducer is: $O(apr) = \{30\}; O(aug) = \{31\}; O(dec) = \{31\}; O(feb) = \{28,29\}; O(jan) = \{31\}; O(jul) = \{31\}$. This transducer is minimal except for the word jul.

Proposition 1. *A subsequential transducer which is minimal except for the empty word ε is minimal.*

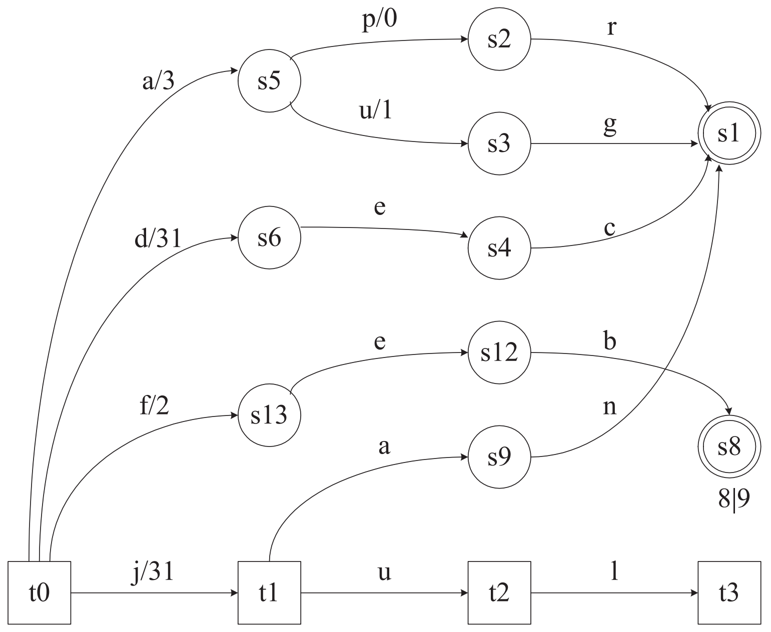

Fig. 1. Subsequential transducer minimal except for jul.

Lemma 1. *Let the subsequential transducer $\mathcal{T} = \langle \Sigma, \Delta, S, s, F, \mu, \lambda, \Psi \rangle$ be minimal except for $\omega = w_1 w_2 \ldots w_k$, $\omega \neq \varepsilon$. Let there be no state equivalent to t_k in the set $S \setminus T$. Then $\mathcal{T}$ is also minimal except for the word $\omega' = w_1 w_2 \ldots w_{k-1}$.*

Lemma 2. *Let the subsequential transducer $\mathcal{T} = \langle \Sigma, \Delta, S, s, F, \mu, \lambda, \Psi \rangle$ be minimal except for $\omega = w_1 w_2 \ldots w_k$, $\omega \neq \varepsilon$. Let the state $p \in S \setminus T$ be equivalent to the state t_k. Then the transducer:*
$\mathcal{T}' = \left\langle \Sigma, \Delta, S \setminus \{t_k\}, s, F \setminus \{t_k\}, \mu', \lambda|_{S\setminus\{t_k\}\times\Sigma}, \Psi|_{S\setminus\{t_k\}} \right\rangle$ *where:*

$$\mu'(r,a) = \begin{cases} \mu(r,a) & \text{, in case } r \neq t_{k-1} \vee a \neq w_k \text{ and } \mu(r,a) \text{ is defined} \\ p & \text{, in case } r = t_{k-1}, a = w_k \\ \text{not defined otherwise} & \end{cases}$$

is equivalent to the transducer $\mathcal{T}$ and is minimal except for the word $\omega' = w_1 w_2 \ldots w_{k-1}$.

Lemma 3. *Let the subsequential transducer $\mathcal{T} = \langle \Sigma, \Delta, S, s, F, \mu, \lambda, \Psi \rangle$ be minimal except for $\omega = w_1 w_2 \ldots w_k$. Then for t_k holds the following statement:*

$$t_k \text{ is equivalent to } r \in S \setminus T \leftrightarrow$$
$$((t_k \in F \leftrightarrow r \in F) \;\&\; (t_k \in F \rightarrow \Psi(t_k) = \Psi(r)) \;\&$$
$$\forall a \in \Sigma \; ((\neg !\mu(t_k, a) \;\&\; \neg !\mu(r, a)) \vee (!\mu(t_k, a) \;\&\; !\mu(r, a) \;\&$$

$$\mu(t_k, a) = \mu(r, a) \ \& \ \lambda(t_k, a) = \lambda(r, a))).$$

Theorem 3. *Let the subsequential transducer* $\mathcal{T} = \langle \Sigma, \Delta, S, s, F, \mu, \lambda, \Psi \rangle$ *be minimal except for* $\omega' = w_1 w_2 \dots w_m$. *Let* $\psi \in L(\mathcal{T})$ *be the last word in the lexicographical order of the input language of the transducer. Let* ω *be a word which is greater in lexicographical order than* ψ. *Let the* $\tau \in \Delta^*$ *be the output for* ω. *Let* ω' *be the longest common prefix of* ψ *and* ω. *In that case we can denote* $\omega = w_1 w_2 \dots w_m w_{m+1} \dots w_k; k > m$. *Let us use* W_n *to denote the word* $W_n = w_1 w_2 \dots w_n$; $n = 1, 2, \dots, k$ *and* $W_0 = \varepsilon$. *Let us use* Λ_n *to denote the word* $\Lambda_n = \lambda^*(t_0, W_n) \wedge \tau$. *Let us define the subsequential transducer* $\mathcal{T}' = \langle \Sigma, S', s, F', \mu', \lambda', \Psi' \rangle$ *as follows:*

$$t_{m+1}, t_{m+2}, \dots, t_k \textit{ are new states such that } S \cap \{t_{m+1}, t_{m+2}, \dots, t_k\} = \emptyset$$

$$S' = S \cup \{t_{m+1}, t_{m+2}, \dots, t_k\}$$

$$F' = F \cup \{t_k\}$$

$$\mu'(r, a) = \begin{cases} t_{i+1} & \textit{, in case } r = t_i, m \le i \le k-1, a = w_{i+1} \\ \mu(r, a) & \textit{, in case } r \in S \textit{ and } \mu(r, a) \textit{ is defined and} \\ & r \ne t_m \vee a \ne w_{m+1} \\ \textit{is not defined otherwise} & \end{cases}$$

$$\lambda'(r, a) = \begin{cases} \lambda(r, a) & \textit{, in case } r = S \setminus \{t_0, t_1, \dots, t_k\} \\ & \vee (r = t_0 \ \& \ a \ne w_1) \\ [\Lambda_{n-1}]^{-1} \Lambda_n & \textit{, in case } r = t_{n-1}, a = w_n, n = 1, 2, \dots, m \\ [\Lambda_n]^{-1} \lambda^*(t_0, W_n a) & \textit{, in case } r = t_n, a \ne w_{n+1}, n = 1, 2, \dots, m \\ [\Lambda_m]^{-1} \tau & \textit{, in case } r = t_m, a = w_{m+1} \\ \varepsilon & \textit{, in case } r = t_n, a = w_{n+1}, n = m+1, \dots, k-1 \\ \textit{is not defined} & \textit{otherwise} \end{cases}$$

$$\Psi'(r) = \begin{cases} \Psi(r) & \textit{, in case } r \notin \{t_1, t_2, \dots, t_k\} \ \& \ r \in F \\ \{\varepsilon\} & \textit{, in case } r = t_k \\ [\Lambda_n]^{-1} \lambda^*(t_0, W_n) \cdot \Psi(r) & \textit{, in case } r = t_n \in F, n = 1, 2, \dots, m \\ \textit{is not defined} & \textit{otherwise} \end{cases}$$

Then the subsequential transducer $\mathcal{T}'$ *is minimal except for* ω, *and the following holds:* $L(\mathcal{T}') = L(\mathcal{T}) \cup \{\omega\}$, $O_{\mathcal{T}'}|_{L(\mathcal{T})} = O_{\mathcal{T}}$ *and* $O_{\mathcal{T}'}(\omega) = \{\tau\}$.

Theorem 4. *Let the subsequential transducer* $\mathcal{T} = \langle \Sigma, \Delta, S, s, F, \mu, \lambda, \Psi \rangle$ *be minimal except for* $\omega = w_1 w_2 \dots w_k$ *and* $\omega \in L(\mathcal{T})$ *be the last word in the lexicographical order of the input language of the transducer. Let* $\tau \in \Delta^*$ *be a new output for* ω, *such that* $\tau \notin O_{\mathcal{T}}(\omega)$. *Let us use* W_n *to denote the word* $W_n = w_1 w_2 \dots w_n$; $n = 1, 2, \dots, k$ *and* $W_0 = \varepsilon$. *Let us use* Λ_n *to denote the word* $\Lambda_n = \lambda^*(t_0, W_n) \wedge \tau$. *Let us define the subsequential transducer* $\mathcal{T}' = \langle \Sigma, S, s, F, \mu, \lambda', \Psi' \rangle$ *as follows:*

$$\lambda'(r, a) = \begin{cases} \lambda(r, a) & \textit{, in case } r = S \setminus \{t_0, \dots, t_k\} \vee (r = t_0 \ \& \ a \ne w_1) \\ [\Lambda_{n-1}]^{-1} \Lambda_n & \textit{, in case } r = t_{n-1}, a = w_n, n = 1, 2, \dots, k \\ [\Lambda_n]^{-1} \lambda^*(t_0, W_n a) & \textit{, in case } r = t_n, a \ne w_{n+1}, n = 1, 2, \dots, k-1 \\ \textit{is not defined} & \textit{otherwise} \end{cases}$$

$$\Psi'(r) = \begin{cases} \Psi(r) & \text{, \textit{in case} } r \notin \{t_1, t_2, \ldots, t_k\} \ \& \ r \in F \\ [\Lambda_n]^{-1}\lambda^*(t_0, W_n) \cdot \Psi(t_n) & \text{, \textit{in case} } r = t_n \in F, n = 1, 2, \ldots, k-1 \\ [\Lambda_k]^{-1}\lambda^*(t_0, W_k) \cdot \Psi(t_k) \cup & \\ \cup\{[\Lambda_k]^{-1}\tau\} & \text{, \textit{in case} } r = t_k \\ \textit{is not defined} & \textit{otherwise} \end{cases}$$

Then the subsequential transducer $\mathcal{T}'$ is minimal except for ω, and the following holds: $L(\mathcal{T}') = L(\mathcal{T})$, $O_{\mathcal{T}'}|_{L(\mathcal{T})\setminus\{\omega\}} = O_{\mathcal{T}}|_{L(\mathcal{T})\setminus\{\omega\}}$ and $O_{\mathcal{T}'}(\omega) = O_{\mathcal{T}}(\omega) \cup \{\tau\}$.

We can use the proving schema introduced in [4] to prove the lemmae and theorems for minimal except for a word subsequential transducer. The only difference is that we have to check that the resulting transducers are canonical.

We can use the following equations for an efficient computation of the functions λ' and ψ' for the last two theorems.

$$\left\langle c_1 = \lambda(t_0, w_1) \wedge \tau \ \ , \ \ l_1 = c_1^{-1}\lambda(t_0, w_1) \ \ , \ \ \tau_1 = c_1^{-1}\tau \right\rangle$$

$$\left\langle c_2 = (l_1\lambda(t_1, w_2)) \wedge \tau_1 \ \ , \ \ l_2 = c_2^{-1}(l_1\lambda(t_1, w_2)) \ \ , \ \ \tau_2 = c_2^{-1}\tau_1 \right\rangle$$

$$\left\langle c_3 = (l_2\lambda(t_2, w_3)) \wedge \tau_2 \ \ , \ \ l_3 = c_3^{-1}(l_2\lambda(t_2, w_3)) \ \ , \ \ \tau_3 = c_3^{-1}\tau_2 \right\rangle$$

$$\vdots$$

$$\left\langle c_m = (l_{m-1}\lambda(t_{m-1}, w_m)) \wedge \tau_{m-1}, l_m = c_m^{-1}(l_{m-1}\lambda(t_{m-1}, w_m)), \tau_m = c_m^{-1}\tau_{m-1} \right\rangle$$

We can calulate c_n, l_n, τ_n iteratively for $n = 1, 2, \ldots, m$.

We can proove by induction that:

$$c_n = [\lambda^*(t_0, W_{n-1}) \wedge \tau]^{-1}(\lambda^*(t_0, W_n) \wedge \tau)$$

$$l_n = [\lambda^*(t_0, W_n) \wedge \tau]^{-1}\lambda^*(t_0, W_n)$$

$$\tau_n = [\lambda^*(t_0, W_n) \wedge \tau]^{-1}\tau$$

for $n = 1, 2, \ldots, m$. Hence we have that:

$$\lambda'(t_{n-1}, w_n) = c_n$$

$$\lambda'(t_n, a) = l_n\lambda(t_n, a)$$

$$\Psi'(t_n) = l_n \cdot \Psi(t_n)$$

for $a \neq w_{n+1}, n = 1, 2, \ldots, m$, and

$$\lambda'(t_m, w_{m+1}) = \tau_m \text{ for Theorem 3, or}$$

$$\Psi'(t_k) = l_k \cdot \Psi(t_k) \text{ for Theorem 4.}$$

Now we can proceed with the description of our method for direct building of minimal subsequential transducer for a given sorted list of words.

Let a non-empty finite list of words L in lexicographical order be given. Let for every word in L the corresponding output is given. Let $\omega^{(i)}$ denotes the i-th word of the list and $\tau^{(i)}$ denotes the output of the i-th word. We start with the minimal canonical subsequential transducer which recognizes only the first word of the list and outputs the output for the first word. This transducer can be built trivially and is also minimal except for $\omega^{(1)}$. Using it as a basis we carry out an induction on the words of the list. Let us assume that the transducer $\mathcal{T}^{(n)}$ with language $L^{(n)} = \{\omega^{(i)} \mid i = 1, 2, \ldots, n\}$ has been built and that $\mathcal{A}^{(n)}$ is minimal except for $\omega^{(n)}$ and $O_{\mathcal{T}^{(n)}}(\omega^{(i)}) = \tau^{(i)}$ for $i = 1, 2, \ldots, n$. We have to build the Transducer $\mathcal{T}^{(n+1)}$ with language $L^{(n+1)} = \{\omega^{(i)} \mid i = 1, 2, \ldots, n+1\}$ which is minimal except for $\omega^{(n+1)}$ and $O_{\mathcal{T}^{(n+1)}}(\omega^{(i)}) = \tau^{(i)}$ for $i = 1, 2, \ldots, n+1$.

Let ω' be the longest common prefix of the words $\omega^{(n)}$ and $\omega^{(n+1)}$. Using several times Lemma 1 and Lemma 2 (corresponding to the actual case) we build the transducer $\mathcal{T}'$ which is equivalent to $\mathcal{T}^{(n)}$ and is minimal except for ω'. Now we can use Theorem 3 (or Theorem 4 if $\omega^{(n)} = \omega^{(n+1)}$) to build the transducer $\mathcal{T}^{(n+1)}$ with language $L^{(n+1)} = L^{(n)} \cup \{\omega^{(n+1)}\} = \{\omega^{(i)} \mid i = 1, 2, \ldots, n+1\}$ which is minimal except for $\omega^{(n+1)}$ and $O_{\mathcal{T}^{(n+1)}}(\omega^{(i)}) = \tau^{(i)}$ for $i = 1, 2, \ldots, n+1$.

In this way by induction we build the minimal except for the last word of the list transducer with language the list L and the given output. At the end using again Lemma 1 and Lemma 2 we build the transducer equivalent to the former one which is minimal except for the empty word. From Proposition 1 we have that it is the minimal subsequential transducer for the list L and corresponding output.

To distinguish efficiently between Lemma 1 and Lemma 2 we can use the condition given in Lemma 3. □

Example 2. Let us consider the following example. On Figure 1 the transducer minimal except jul with input language $\{apr, aug, dec, feb, jan, jul\}$ and output function $O(apr) = \{30\}$; $O(aug) = \{31\}$; $O(dec) = \{31\}$; $O(feb) = \{28, 29\}$; $O(jan) = \{31\}$; $O(jul) = \{31\}$ is given. After the application of Lemma 2 and Theorem 3 we will construct the transducer minimal except for jun where $O(jun) = \{30\}$. This transducer is given on Figure 2. In this way we are adding the next word with the corresponding output to the transducer.

3 Algorithm for building of minimal subsequential transducer for a given sorted list

Here we give the pseudo-code in a Pascal-like language (like the language used in [1]). We will presume that there are given implementations for Abstract Data Types (ADT) representing transducer state and dictionary of transducer states. Later we presume that NULL is the null constant for arbitrary abstract data type.

On Transducer state we will need the following types and operations:

1. STATE is pointer to a structure representing a transducer state;
2. FIRST_CHAR, LAST_CHAR : are the first and the last char in the input alphabet;
3. **function** NEW_STATE : STATE returns a new state;

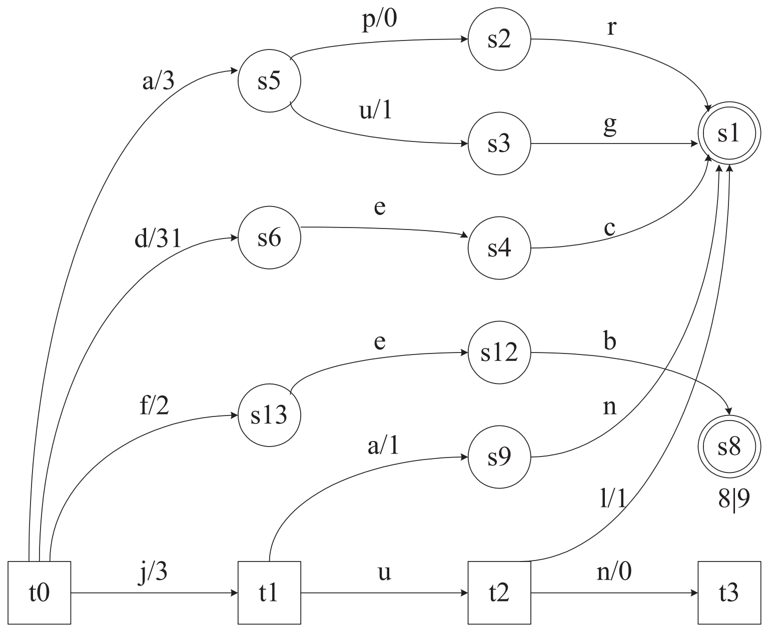

Fig. 2. Subsequential transducer minimal except for *jun*.

4. **function** FINAL(STATE) : boolean returns true if the state is final and false otherwise;
5. **procedure** SET_FINAL(STATE, boolean) sets the finality of the state to the boolean parameter;
6. **function** TRANSITION(STATE, char) : STATE returns the state to which the transducer transits from the parameter state with the parameter char;
7. **procedure** SET_TRANSITION(STATE, char, STATE) that sets the transition from first parameter state by the parameter char to the second parameter state;
8. **function** STATE_OUTPUT(STATE) : **set of** string returns the output set of strings on final states;
9. **procedure** SET_STATE_OUTPUT(STATE, **set of** string) sets the output set of strings on final states;
10. **function** OUTPUT(STATE, char) : string returns the output string for the transition from the parameter state by the parameter char;
11. **procedure** SET_OUTPUT(STATE, char, string) sets the output string for the transition from the parameter state by the parameter char;
12. **procedure** PRINT_TRANSDUCER(file, STATE) prints the transducer starting from the parameter state to file.

Having defined the above operations we make use of the following three functions and procedures:

1. **function** COPY_STATE(STATE) : STATE copies a state to a new one;
2. **procedure** CLEAR_STATE(STATE) clears all transitions of the state and sets it to non final;
3. **function** COMPARE_STATES(STATE, STATE) : integer compares two states

The ADT on Dictionary of transducer states uses the COMPARE_STATES function above to compare states. For the dictionary we need the following operations:

1. **function** NEW_DICTIONARY : DICTIONARY returns a new empty dictionary;
2. **function** MEMBER(DICTIONARY,STATE) : STATE returns state in the dictionary equivalent to the parameter state or NULL if not present;
3. **procedure** INSERT(DICTIONARY,STATE) inserts state to dictionary.

Implementation for the above ADTs could be found in e.g. [1]. Now we are ready to present the pseudo-code of our algorithm.

Algorithm 5. *For direct building of minimal subsequential transducer presenting the input list of words given in lexicographical order with their corresponding outputs.*

```
program Create_Minimal_Transducer_for_Given_List ( input, output);
var
    MinimalTransducerStatesDictionary : DICTIONARY;
    TempStates : array [0..MAX_WORD_SIZE] of STATE;
    InitialState : STATE;
    PreviousWord, CurrentWord, CurrentOutput,
          WordSuffix, CommonPrefix : string;
    tempString : string;
    tempSet : set of string;
    i, j, PrefixLengthPlus1 : integer;
     c : char;
     function FindMinimized ( s : STATE) : STATE;
     {returns an equivalent state from the dictionary. If not present –
        inserts a copy of the parameter to the dictionary and returns it.}
     var r : STATE:
     begin
         r := MEMBER(MinimalTransducerStatesDictionary,s);
         if r = NULL then begin
             r := COPY_STATE(s);
             INSERT(r);
         end;
         return(r);
     end; {FindMinimized}
 begin
```

```
23  MinimalTransducerStatesDictionary := NEW_DICTIONARY;
24  for i := 0 to MAX_WORD_SIZE do
25      TempState[i] := NEW_STATE;
26  PreviousWord := '';
27  CLEAR_STATE(TempState[0]);
28  while not eof(input) do begin
29  {Loop for the words in the input list}
30      readln(input,CurrentWord,CurrentOutput);
31      { the following loop calculates the length of the longest common
          prefix of CurrentWord and PreviousWord }
32      i := 1;
33      while (i<length(CurrentWord)) and (i<length(PreviousWord))
              and (PreviousWord[i] = CurrentWord[i]) do
34          i := i+1;
35      PrefixLengthPlus1 := i;
36      {we minimize the states from the suffix of the previous word }
37      for i := length(PreviousWord) downto PrefixLengthPlus1 do
38          SET_TRANSITION(TempStates[i-1],PreviousWord[i],
                    FindMinimized(TempStates[i]));
39      {This loop initializes the tail states for the current word}
40      for i := PrefixLengthPlus1 to length(CurrentWord) do begin
41          CLEAR_STATE(TempStates[i]);
42          SET_TRANSITION(TempStates[i-1],CurrentWord[i],
                    TempStates[i]);
43      end;
44      if CurrentWords <> PreviousWord then begin
45          SET_FINAL(TempStates[length(CurrentWord)], true);
46          SET_OUTPUT(TempStates[length(CurrentWord)], {''});
47      end;
48      for j := 1 to PrefixLengthPlus1-1 do begin
49          CommonPrefix := OUTPUT(TempStates[j-1], CurrentWord[j])
                    $\wedge$ CurrentOutput;
50          WordSuffix := CommonPrefix$^{-1}$ OUTPUT(TempStates[j-1],
                    CurrentWord[j]);
51          SET_OUTPUT(TempStates[j-1], CurrentWord[j],
                    CommonPrefix);
52          for c := FIRST_CHAR to LAST_CHAR do begin
53              if TRANSITION(TempStates[j],c) <> NULL then
54                  SET_OUTPUT(TempStates[j],c,concat(WordSuffix,
                            OUTPUT(TempStates[j],c)));
55          end;
56          if FINAL(TempStates[j]) then begin
57              tempSet := $\emptyset$;
58              for tempString in STATE_OUTPUT(TempStates[j]) do
59                  tempSet := tempSet $\cup$ concat(WordSuffix,tempString);
```

```
                SET_STATE_OUTPUT(TempStates[j], tempSet);
            end;
            CurrentOutput := CommonPrefix^-1 CurrentOutput;
        end;
        if CurrentWord = PreviousWord then
            SET_STATE_OUTPUT(TempStates[length(CurrentWord)],
                 STATE_OUTPUT(TempStates[length(CurrentWord)])
                 ∪ CurrentOutput);
        else SET_OUTPUT(TempStates[PrefixLengthPlus1-1],
                     CurrentWord[PrefixLengthPlus1],CurrentOutput);
        PreviousWord := CurrentWord;
    end; {while}
    { here we are minimizing the states of the last word }
    for i := length(CurrentWord) downto 1 do
        SET_TRANSITION(TempStates[i-1],PreviousWord[i],
                 FindMinimized(TempStates[i]));
    InitialState := FindMinimized(TempStates[0]);
    PRINT_TRANSDUCER(output,InitialState);
end.
```

4 Implementation results and comparisons

Based on the main algorithm for direct building of minimal automata we have created implementation for direct construction of minimal automaton with labeled final states and minimal subsequential transducer. The results are summarized in the table bellow. We used a Bulgarian grammatical dictionary of simple words with about 900000 entries for the experiments. An implementation of the algorithm given in [3] has been used for the construction of the pseudo-minimal subsequential transducer.

In [6] Mehryar Mohri reports that the construction with his method of the p-subsequential transducer for a 672000 entries French dictionary takes 20' on a HP/9000 755 computer. All our experiments have been performed on a 500MHz Pentium III personal computer with 128MB RAM.

References

1. A. Aho, J. Hopcroft, J. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, Massachutes, 1983.
2. J. Daciuk, S. Mihov, B. Watson, R. Watson, Incremental Construction of Minimal Acyclic Finite State Automata, *Computational Linguistics*, Vol. 26(1), 2000.
3. D. Maurel, Pseudo-minimal transducer, *Theoretical Computer Science*, Vol. 231(1), 129-139, 2000.
4. S. Mihov, Direct Building of Minimal Automaton for Given List, *Annuaire de l'Université de Sofia "St. Kl. Ohridski"*, Faculté de Mathématique et Informatique, volume 91, livre 1, 1998.

Table 1. Comparison between different automata for the representation of a large-scale grammatical dictionary for Bulgarian.

Number of lines	895453		
Initial size	27.5 MB		
	Minimal Transducer	Pseudo-minimal Transducer	Minimal automaton with labeled final states
States	43413	531397	47854
Transitions	106809	992412	110791
Codes	16378	16378	6016
p	5	–	–
Size of codes	209K	–	126K
Size of automaton	1.3M	–	800K
Construction time	2'35"	–	25"
Memory used	5M	108M	2.5M

5. M. Mohri, Minimization of Sequential Transducers, *Lecture Notes in Computer Science*, Springer, Berlin, 1994.
6. M. Mohri, On Some Applications of Finite-State Automata Theory to Natural Language Processing, *Natural Language Engineering*, Vol. 2(1), 1-20, 1996.
7. M. Mohri, Finite-State Transducers in Language and Speech Processing, *Computational Linguistics*, Vol. 23(2), 269-311, 1997.
8. D. Revuz, *Dictionaires et lexiques – Méthodes et algorithmes*, Doctoral dissertation in Computer Science, University Paris VII, Paris, 1991.