Finite-state transducers play an important role in natural language processing. They provide a model for text and speech, as well as transformations of them. In order to reduce the complexity of their application, deterministic and minimal transducers are required. This essay introduces a particular type of finite-state transducers, subsequential string-to-weight transducers, and presents algorithms for their determinization and minimization that have been proposed by Mehryar Mohri. These algorithms are compared with their classical counterparts for finite-state machines and illustrated by examples.

1 Introduction

Natural language processing requires models to represent information about text and speech, as well as rules describing their recognition. Finite-state machines provide such a model and can be widely applied in this field.

One advantage of finite-state machines is their moderate time and space complexity under certain conditions. Deterministic finite-state machines provide a time complexity that is linear to the input size. Space complexity can be reduced by minimization of the machines. Algorithms for determination and minimization of finite-state machines have been proposed and successfully applied, e.g. in compiler construction.

Often, natural language processing not only requires finite-state machines to recognize text and speech, but to transform one representation into another. Furthermore, it is desirable to assign weights to the output of these transformations. For instance, when mapping sequences of words to sentences, one wants to know which of the possible sentences is most likely. In addition to recognizing text and speech, finite-state machines can be extended such that they produce strings or weights as output.

This extension leads to transducers, depending on the type of input and output for instance to string-to-weight transducers. Frequently, the weights are interpreted as negative logarithms of probabilities. Thus, the minimum output corresponds to the highest probability. In order to benefit from moderate complexities, determinization and minimization algorithms for transducers are needed. This text will present these algorithms based on the work of Mehryar Mohri in [Moh97].

The remainder of this essay is organized as follows. At first, finite-state machines and algorithms to obtain deterministic and minimum machines are presented. Afterwards, subsequential string-to-weight transducers and their relation to formal power series are described. Algorithms for determination and minimization are explained and illustrated with examples. Finally, it is shown how transducers can be applied in speech recognition.

2 Finite-State Machines

This section introduces finite-state machines and algorithms for their determinization and minimization. These are proposed in [ASU86], that
uses finite-state machines as recognizers for languages during the lexical analysis of a compiler. A finite-state machine, FSM, (also called finite automaton) is a mathematical model of behavior consisting of states and state transitions. It can be deterministic or nondeterministic. Nondeterministic means that more than one transition from a state may be possible on one input symbol. Both types of FSMs are capable of recognizing exactly the same languages, namely regular languages. In contrast, they differ regarding their time and space complexity. Nondeterministic machines potentially have to backtrack in case the input cannot be recognized, so as to test all possible state transitions. Since deterministic machines allow only one transition from a state on each input, no backtracking is required. Thus, they consume less processing time. However, nondeterministic machines can be much smaller in terms of the number of states than equivalent deterministic ones. Hence, there is a time-space tradeoff between deterministic and nondeterministic FSMs.

In this essay, nondeterministic finite-state machines are defined as follows ($\epsilon$ denotes the empty word).

**Definition: Nondeterministic FSM**
A nondeterministic FSM is a mathematical model presented by a 5-tuple $(Q, \Sigma, \delta, i, F)$ with:

- a set of states: $Q$
- a set of input symbols, i.e. an input alphabet: $\Sigma$
- a transition function that maps state-symbol pairs to sets of states: $\delta : Q \times \Sigma \cup \{\epsilon\} \to \mathcal{P}(Q)$
- an initial state: $i \in Q$
- a set of accepting states, i.e. final states: $F \subseteq Q$

Figure 1 is an example of a nondeterministic finite-state machine with five states. The nondeterminism is caused, for example, by $\delta(1, a) = \{1, 2\}$.

### 2.1 Determinization
A deterministic FSM has particular properties. At first, there are no $\epsilon$-transitions, i.e. transitions on input $\epsilon$. Next, for each state $s$ and input symbol $a$, there is at most one edge labeled $a$ leaving $s$.

**Definition: Deterministic FSM**
A deterministic FSM is a nondeterministic FSM in which the transition function is defined as $\delta : Q \times \Sigma \to Q$.

An example is given in Figure 2. The machine is equivalent to the one in Figure 1, i.e. it accepts exactly the same inputs. In the following, an algorithm to convert a nondeterministic FSM into an equivalent deterministic one is presented. It is known as subset construction.

The algorithm uses three operations on states of a nondeterministic FSM whose exact computation is omitted here, but that can be found in [ASU86]. Let $s$ be a state of a nondeterministic FSM, and let $T$ be a set of such states.

- $\epsilon$-closure($s$): Set of state $s$ and states reachable from $s$ on $\epsilon$-transitions only.
- $\epsilon$-closure($T$): Set of states in $T$ and states reachable from some $s \in T$ on $\epsilon$-transitions only.
- move($T$, $a$): Set of states to which there exists a transition on input symbol $a$ from some $s \in T$. 

![Figure 1: Example of a nondeterministic FSM.](image1)

![Figure 2: Example of a deterministic FSM.](image2)
The algorithm transforms a nondeterministic FSM $N = (Q_1, \Sigma, \delta_1, i_1, F_1)$ into a deterministic FSM $D = (Q_2, \Sigma, \delta_2, i_2, F_2)$. The main idea is to merge multiple states from $N$, reachable from one state by the same input symbol, into one state in $D$. The algorithm builds a new transition function $\delta_2$ such that all transitions from $N$ are simulated in $D$.

Figure 3 presents the subset construction, which defines the transition function $\delta_2$. The new initial state $i_2$ is $\epsilon$-closure($i_1$). Each state $q_2$ represents a set of states from $N$. If at least one element in that set is a final state in $N$, the corresponding state in $D$ is also final. Note that in line 4, input symbols without any outgoing transition from states in $q_2$, i.e. yielding $U = \emptyset$, are ignored.

The FSM from Figure 2 is the result of the determinization algorithm applied to the machine from Figure 1. The values of important variables during the algorithm’s execution is shown in Figure 4.

$\epsilon$-closure($i_1$) consists of states 0 and 1, since 1 is reachable from the initial state 0 on input $\epsilon$. Thus, $\{0, 1\}$ is the initial state of $D$ (step 1). It is the only unmarked state in $Q_2$, such that in the first (outer) iteration $q_2 = \{0, 1\}$. On input $a$, 2 is reachable from 0, and 1 and 2 are reachable from 1. Hence in $D$, the new state $\{1, 2\}$ is reachable from $\{0, 1\}$ on input $a$ (step 2). The algorithm continues until all states from $Q_2$ are marked, i.e. the transition function of $D$ is completely defined.

2.2 Minimization

Since determinization of FSMs may lead to large machines having many states, it is desirable to minimize them. It can be shown that each regular set is recognized by a deterministic FSM with minimal number of states that is unique up to state names. In this section, the minimization algorithm proposed in [ASU86] is shortly sketched.

**Definition: Minimal deterministic FSM**

A deterministic FSM is minimal, if there exists no equivalent deterministic FSM, i.e. accepting the same set of input strings, with less states.

The main idea of minimization is to partition the set of states into equivalence classes and, based on this partition, distinguish states. Two states 1 and 2 are distinguished if one is final and the other nonfinal. Furthermore, they are distinguished by input $w$ if there is a path with input $w$ from 1 to a state in one equivalence class, while starting from 2 with input $w$ leads to a state in another equivalence class. The algorithm works on a partition of states that is refined step by step. The initial partition consists of two equivalence classes: accepting and nonaccepting states. The essential refinement step takes a group of states and an input symbol, and checks whether the states of the group are distinguished by that input symbol. If so, they have to be put into different equivalence classes. This step is repeated until no more groups can be splitted. After minimization, all indistinguishable states are merged.

A complete description of the algorithm can be found in [ASU86] and is omitted here.

3 Subsequential

String-to-Weight Transducers

This section introduces a particular kind of finite-state transducers introduced by Mohri in [Moh97]: subsequential string-to-weight transducers. At first, a number of terms are defined and illustrated with examples. Secondly, the determinization algorithm for classical FSMs is extended towards subsequential string-to-weight transducers. Finally, an adapted minimization algorithm is presented.

3.1 General Transducers

FSMs are classified into acceptors/recognizers and transducers. Acceptors produce a binary output (yes or no) that indicates whether a given input is accepted by the FSM or not. Transducers extend this idea by creating strings or weights as output. Depending on the type of accepted input and produced output, one speaks e.g. of string-to-string transducers or string-to-weight transducers.

Transducers may be deterministic or nondeterministic regarding their input. A transducer with a deterministic input is called sequential transducer. That is, for each state there exists at most one outgoing transition on one input symbol. However, sequential transducers may have nondeterministic output. Thus, multiple outgoing transitions with one output symbol may oc-
initially, $\epsilon$-closure($i_1$) is the only state in $Q_2$ and it is unmarked

while there is an unmarked state $q_2$ in $Q_2$ do begin
    mark $q_2$
    for each input symbol $a$ do begin
        $U := \epsilon$-closure(move($q_2$, $a$))
        if $U$ is not in $Q_2$ then
            add $U$ as an unmarked state to $Q_2$
        $\delta_2(q_2, a) := U$
    end
end

Figure 3: The subset construction.

<table>
<thead>
<tr>
<th>Step</th>
<th>Unmarked states</th>
<th>$q_2$</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{0, 1}</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>{0, 1}</td>
<td>$\delta_2({0, 1}, a) := {1, 2}$</td>
</tr>
<tr>
<td>3</td>
<td>{1, 2}</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>{1, 2}</td>
<td>$\delta_2({1, 2}, a) := {1, 2}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\delta_2({1, 2}, b) := {3}$</td>
</tr>
<tr>
<td>5</td>
<td>{3}</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>{3}</td>
<td>$\delta_2({3}, a) := {4}$</td>
</tr>
<tr>
<td>7</td>
<td>{4}</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>{4}</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 4: Step-by-step execution of determinization algorithm for a finite-state machine.

Sequential transducers can be extended such that whenever the transducer ends up in a final state additional output is generated. Such devices are called subsequential transducers.

3.2 String-to-Weight Transducers

The rest of this essay focuses on string-to-weight transducers, their properties and algorithms applied to them. In addition to output weights, they provide initial weights added to initial states. String-to-weight transducers are also called weighted automata. For instance, the transducer from Figure 6 has initial weight 4, and output weight 1 in its final state 2.

The output produced by a recognized input string is a combination of the initial weight, the output weight of all used transitions, and the final output weight. In most applications, it is simply the sum of all involved weights. For example, on input $ab$ the transducer from Figure 6 produces $4 + 1 + 3 + 1 = 9$.

Formally, string-to-weight transducers can be defined as follows.
Definition: String-to-weight transducer
A string-to-weight transducer $T$ is a 7-tuple $T = (Q, \Sigma, I, F, E, \lambda, \rho)$ with:
- a finite set of states: $Q$
- the input alphabet: $\Sigma$
- the set of initial states: $I \subseteq Q$
- the set of final states: $F \subseteq Q$
- a finite set of transitions: $E \subseteq Q \times \Sigma \times \mathbb{R}_+ \times Q$
- the initial weight function: $\lambda : I \rightarrow \mathbb{R}_+$
- the final weight function: $\rho : F \rightarrow \mathbb{R}_+$

Similarly to FSMs, one can define a transition function $\delta$ mapping $Q \times \Sigma$ to $\mathcal{P}(Q)$:

$$\forall (q, a) \in Q \times \Sigma, \quad \delta(q, a) = \{ q' \mid \exists x \in \mathbb{R}_+ : (q, a, x, q') \in E \}$$

In addition, one can derive an output function $\sigma$ assigning to each transition its weight:

$$\forall t = (p, a, x, q) \in E, \sigma(t) = x$$

A path $\pi$ is a sequence of successive transitions from one state to another:

$$\pi = (q_0, a_0, x_0, q_1), \ldots, (q_{m-1}, a_{m-1}, x_{m-1}, q_m)$$

with $(q_i, a_i, x_i, q_{i+1}) \in E$. The label of a path $\pi$ is $a_0 \ldots a_{m-1}$.

The set of paths from $q$ to $q'$ labeled with input string $w$ is abbreviated by:

$$q \xrightarrow{w} q'$$

Using the notion of a path, the transition function can be extended from single input symbols to input strings by:

$$\forall (q, w) \in (Q \times \Sigma^*), \delta(q, w) = \{ q' : \exists \pi, x \in q \xrightarrow{w} q' \}$$

The output function is extended to paths by:

$$\sigma(\pi) = \sum_{0 \leq i \leq m-1} x_i$$

where $x_i$ are the output weights of the $m$ transitions constituting the path.

Now, we combine string-to-weight transducers with the subsequential property mentioned above. The result are subsequential string-to-weight transducers.

Definition: Subsequential string-to-weight transducer
A subsequential string-to-weight transducer is a 8-tuple $(Q, i, F, \Sigma, \delta, \sigma, \lambda, \rho)$ with:
- a finite set of states: $Q$
- an initial state: $i \in Q$
- the set of final states: $F \subseteq Q$
- the input alphabet: $\Sigma$
- the transition function:
  $$\delta : (Q \times \Sigma) \rightarrow Q$$
- the output function:
  $$\sigma : (Q \times \Sigma^*) \rightarrow \mathbb{R}_+$$
- the initial weight: $\lambda \in \mathbb{R}_+$
- the final weight function: $\rho : F \rightarrow \mathbb{R}_+$

The extension of the output function to input strings is defined as follows:

$$\sigma(q, w) = \sigma(\pi)$$

where $\pi$ is a path with label $w \in \Sigma^*$ starting in state $q \in Q$.

3.3 Relation to Formal Power Series

String-to-weight transducers compute functions that map $\Sigma^*$ to $\mathbb{R}_+$. This allows to view them as formal power series, mathematical devices related to usual power series. In numerous applications of string-to-weight transducers, two operations are used: Addition, to combine all weights on a path, and min, since usually, only the minimum of the output paths for a given input string is interesting. Thus, the following semiring, called the tropical semiring, can be used:

$$\mathcal{R}_+ \cup \{\infty\}, \min, +, \infty, 0$$

A formal power series is rational if and only if it is recognizable, i.e. realizable by a string-to-
weight transducer. Thus, the functions considered in this work are rational power series over the tropical semiring. 

Interpreting formal power series as functions allows to use the following terminology to relate them to transducers:

- The image by a formal power series $S$ of a string $w$ is denoted by $(S, w)$ and called the coefficient of $w$ in $S$.

- The support of $S$ is the language defined by: $\text{supp}(S) = \{ w \in \Sigma^* : (S, w) \neq \infty \}$, i.e. the language accepted by the transducer.

The fact that transducers can be generalized to formal power series over a semiring is used in [Moh97] to propose determinization and minimization algorithms in a very general way. Therefore, they cannot only be applied to string-to-weight transducers using the tropical semiring, but e.g. also to string-to-string transducers using the string semiring.

### 3.4 Determinization of String-to-Weight Transducers

In the following, a determinization algorithm for string-to-weight transducers is presented that is derived from the algorithm proposed in [Moh97]. Whereas the version of Mohri applies to the general case of a semiring $(\mathbb{S}, \oplus, \odot, 0, 1)$, here, the algorithm is presented specifically for the tropical semiring $(\mathbb{R}_+ \cup \{\infty\}, \text{min}, +, \infty, 0)$. This enhances comprehensibility since the focus of this work is on string-to-weight transducers. It will be compared with the powerset construction from Section 2.1 and illustrated by examples.

The determinization algorithm takes a non-subsequential string-to-weight transducer $T_1 = (Q_1, \Sigma, I_1, F_1, E_1, \lambda_1, \rho_1)$ and transforms it into an equivalent, subsequential string-to-weight transducer $T_2 = (Q_2, I_2, F_2, \Sigma, \delta_2, \sigma_2, \lambda_2, \rho_2)$. Its main steps are very similar to the subset construction. Multiple states reachable from one state on the same input symbol are merged. The algorithm begins with the initial states of $T_1$ and builds new states for $T_2$ by iterating over non-marked states.

However, due to the extension of string-to-weight transducers compared with FSMs, the determinization algorithm has to be extended as well. E.g. multiple transitions having the same input label may produce different outputs. When merging states, only the minimum output is considered, whereas the residual weights have to be stored. Hence, the states $q_2$ are not simply sets of states $q_1$ but sets of pairs $(q_1, x)$, where $x$ stores the residual weight not considered on the transition leading to the state $q_2$. Furthermore, the algorithm has to deal with input and final weights.

Figure 7 presents the complete algorithm. Similarly to the algorithm from Figure 3, it is build around a set of states $Q_2$ belonging to the resulting transducer. The elements of this set are sets of pairs $(q_1, x)$, where $q_1$ is a state from the input transducer $T_1$. At the beginning, $Q_2$ contains only one element which will be the initial state of $T_2$. It is obtained by taking the minimum weight of all initial states of $T_1$ (line 1). The residual weights of the other initial states $i \in I_1$ are calculated by $\lambda_1(i) - \lambda_2$ and stored at the second position of the corresponding pair (line 2).

The algorithm iterates until all states of $T_2$ are marked. An iteration step dealing with a state $q_2$ consists of two main parts. Firstly, it checks whether $q_2$ contains any final state from $T_1$. If that is the case, $q_2$ is added to the set of final states of $T_2$ (lines 6 and 7). The final weight of $q_2$ is calculated as minimum of the final weights of all included $q_1 \in F_1$, when adding its residual weight in $q_2$ (line 8). The residual weight was stored before when only the minimum weight of a transition was considered. Thus, in order not to omit it, it is added as output weight in the final state.

Secondly, the outgoing transitions of the states included in $q_2$ are considered for each input symbol. The algorithm uses the following notations:

- $\text{dest}(t)$ is the destination state of a transition $t \in E_1$:
  \[ \text{dest}(t) = q' \text{ if } t = (q, a, x, q') \in E_1 \]

- $\text{out}(q_2, a)$ denotes the set of pairs $(q, x)$ contained in $q_2$ that have an outgoing transition $t$ on input $a$:
  \[ \text{out}(q_2, a) = \{(q, x, t) \in q_2 \times E_1 : t = (q, a, \sigma_1(t), \text{dest}(t)) \in E_1\} \]

- $\text{reachable}(q_2, a)$ stands for the set of states reachable by transitions labeled with input $a$.
\[
\lambda_2 := \min_{i \in I_1} \lambda_1(i)
\]
\[
i_2 := \bigcup_{i \in I_1} \{(i, \lambda_1(i) - \lambda_2)\}
\]

Initially, \(i_2\) is the only state in \(Q_2\) and it is unmarked.

while there is an unmarked state \(q_2\) in \(Q_2\) do begin
  mark \(q_2\)
  if there exists \((q, x)\) in \(q_2\) such that \(q \in F_1\) then
    add \(q_2\) to \(F_2\)
    \(\rho_2(q_2) := \min_{q \in F_1, (q, x) \in q_2} (x + \rho_1(q))\)
  for each input symbol \(a\) and transition \(t\) such that \(\text{out}(q_2, a, t) \neq \emptyset\) do begin
    \(\sigma_2(q_2, a) := \min_{(q, x, t) \in \text{out}(q_2, a)} x + \sigma_1(t)\)
    \(\delta_2(q_2, a) := \min_{q' \in \text{reachable}(q_2, a)} \{(q', \min_{(q, x, t) \in \text{out}(q_2, a), \text{dest}(t) = q'} (x + \sigma_1(t) - \delta_2(q_2, a)))\}\)
    if \(\delta_2(q_2, a)\) is not in \(Q_2\) then
      add \(\delta_2(q_2, a)\) as an unmarked state to \(Q_2\)
  end
end

Figure 7: Determinization algorithm for string-to-weight transducers.

from the states contained in \(q_2\):
\[
\text{reachable}(q_2, a) = \{q' : \exists (q, x) \in q_2 : \exists t = (q, a, \sigma_1(t), q') \in E_1\}
\]

Starting from state \(q_2\), the algorithm builds new states for the resulting transducer \(T_2\) by iterating over all input symbols. Only those input symbols on transitions leading out of a state contained in \(q_2\) are considered (line 9). For a given input symbol \(a\), all destination states are merged into one. The weight of the resulting transition is the minimum of the residual weight plus the corresponding transition weight from \(T_1\). The minimum is determined over all possible states included in \(q_2\) and all transitions with input label \(a\) (line 10). The merged destination state contains pairs for all states reachable using input label \(a\). The residual weights are calculated adding the residual weight from the source state and the transition weight in \(T_1\), less the calculated transition weight in \(T_2\) (line 11). If the merged state is not yet in \(Q_2\), it is added as unmarked state (lines 12 and 13).

Unfortunately, the algorithm is not applicable to all nonsubsequential string-to-weight transducers. In some cases, it generates an infinite state space and does not terminate.

Example

In the following, the algorithm is illustrated by an example. The input is the transducer from Figure 8. Obviously, it is nonsubsequential, e.g. because of multiple transitions on input symbol \(a\) leaving state 0.

![Figure 8: Nonsubsequential string-to-weight transducer.](image)

Figure 9 shows all important steps of an execution of the algorithm. An intermediate result is presented in Figure 10. It shows the resulting transducer after step 4 from Figure 9. The initial state containing only the pair \((0, 0)\) is already marked, whereas the other two states have still to be considered by the algorithm.

The final result is presented in Figure 11. Note that only the smallest output for a given input string is kept in the subsequential transducer. For instance, the input \(ab\) admits several outputs in
<table>
<thead>
<tr>
<th>Step</th>
<th>Unmarked states</th>
<th>$q_2$</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>$\lambda_2 := \min(0) = 0$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$i_2 := {(0,0)}$</td>
</tr>
<tr>
<td>2</td>
<td>${(0,0)}$</td>
<td>-</td>
<td>$\sigma_2({(0,0)}, a) := \min(0 + \min(1,3)) = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\delta_2({(0,0)}, a) := {(1, \min(0 + 3 - 1)), (2, \min(0 + 1 - 1))}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>= ${(1,2),(2,0)}$</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>${(0,0)}$</td>
<td>$\sigma_2({(0,0)}, b) := \min(0 + \min(1,4)) = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\delta_2({(0,0)}, b) := {(1, \min(0 + 1 - 1)), (2, \min(0 + 4 - 1))}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>= ${(1,0),(2,3)}$</td>
</tr>
<tr>
<td>4</td>
<td>${(1,2),(2,0)}$, ${(1,0),(2,3)}$</td>
<td>${(1,2),(2,0)}$</td>
<td>add ${(1,2),(2,0)}$ to $F_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\rho_2({(1,2),(2,0)}) := \min(2 + 0) = 2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\sigma_2({(1,2),(2,0)}, b)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>:= $\min(2 + \min(3,5), 0 + \min(1,3)) = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\delta_2({(1,2),(2,0)}, b)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>:= ${(3, \min(2 + 3 - 1, 2 + 5 - 1, 0 + 1 - 1, 0 + 3 - 1))}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>= ${(3,0)}$</td>
</tr>
<tr>
<td>5</td>
<td>${(1,0),(2,3)}$, ${(3,0)}$</td>
<td>${(1,0),(2,3)}$</td>
<td>add ${(1,0),(2,3)}$ to $F_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\rho_2({(1,0),(2,3)}) := \min(0 + 0) = 0$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\sigma_2({(1,0),(2,3)}, b)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>:= $\min(0 + \min(3,5), 3 + \min(1,3)) = 3$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\delta_2({(1,0),(2,3)}, b)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>:= ${(3, \min(0 + 3 - 3, 0 + 5 - 2, 3 + 1 - 3, 3 + 3 - 3))}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>= ${(3,0)}$</td>
</tr>
<tr>
<td>6</td>
<td>${(1,0),(2,3)}$, ${(3,0)}$</td>
<td>${(1,0),(2,3)}$</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>${(3,0)}$</td>
<td>${(1,0),(2,3)}$</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>${(3,0)}$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>${(3,0)}$</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 9: Step-by-step execution of determinization algorithm.

Figure 10: Intermediate result of the determinization algorithm.

Figure 11: Final result of the determinization algorithm.

Remarks

Both space and time complexity of the determinization algorithm are exponential. Similarly
to the algorithm from [ASU86], the resulting transducers may be of exponential size with respect to an equivalent nonsubsequential transducer. However, [Moh97] claims that in some examples that are interesting for speech recognition, the determinization algorithm turns out to be fast, and the resulting transducer has fewer states than the initial one.

As mentioned above, the determinization algorithm is not applicable to all nonsubsequential string-to-weight transducers. [Moh97] gives an algorithm to test whether this is the case, which is not subject of this work.

It is interesting to note that the algorithm described above is equivalent to the one presented in Section 2.1 under certain conditions. When the initial and final weights as well as all output weight are equal to 0, the resulting subsequential transducer is exactly the one obtained by subset construction.

### 3.5 Minimization of String-to-Weight Transducers

Analogously to FSMs, it is helpful to be able to minimize subsequential transducers. [Moh97] defines a minimization algorithm for subsequential power series on the tropical semiring. Furthermore, Mohri proves that there always exists a minimal subsequential transducer, calculates its minimal number of states, and gives a proof for the correctness of the proposed algorithm. Due to limited space, this work solely presents the algorithm and explains it with an example.

Minimization of a string-to-weight transducer consists of two steps: At first, an operation called pushing adapts the transducer such that its output weights are spread differently. Afterwards, the usual minimization algorithm for FSMs is applied.

We define a function $d$ giving the minimum output necessary to reach a final state. For a given subsequential transducer $T = (Q, i, F, \Sigma, \delta, \sigma, \lambda, \rho)$, for any state $q \in Q$, $d$ is defined by:

$$d(q) = \min_{\text{dest}(q,w) \in F} (\sigma(q,w) + \rho(\text{dest}(q,w)))$$

Pushing can now be defined as follows:

**Definition: Pushing**

If $T$ is a subsequential transducer, the result of the pushing operation is a new subsequential transducer $T' = (Q, i, F, \Sigma, \delta, \sigma', \lambda', \rho')$ that only differs from $T$ by its output weights in the following way:

- $\lambda' = \lambda + d(i)$
- $\forall (q,a) \in Q \times \Sigma,
  \sigma'(q,a) = \sigma(q,a) + d(\text{dest}(q,a)) - d(q)$
- $\forall q \in Q, \rho'(q) = 0$

Note that pushing does not change the topology of the transducer. As shown in [Moh97], pushing a subsequential transducer $T$ yields a subsequential transducer $T'$ realizing the same function as $T$.

The minimization algorithm for a subsequential transducer $T$ can be defined quite easily. One has to apply the following two operations one after the other:

1. Pushing
2. FSM minimization

In contrast to FSM minimization, the resulting minimal transducer is not unique up to state names. In general, there are several distinct minimal, i.e. having a minimal number of states, subsequential transducers realizing the same function. They may differ by the way the output weights are spread over the topology. By applying the pushing operation, one can transform several minimal transducers into another.

**Example**

In the following, an example illustrates the minimization of a subsequential string-to-weight transducer. At first, the pushing operation is applied on the input transducer in Figure 12. The new initial weight $\lambda'$ is the sum of the old initial weight $\lambda$ and $d(i)$, i.e. the minimum weight necessary to reach a final state. With $\lambda = 0$ and $d(i) = 1+2+1+2+2 = 0+1+3+2+2 = 8$, the new initial weight $\lambda'$ equals 8. Following the definition of the pushing operation, the output weight is set to 0 for each final state, i.e. $\rho'(6) = 0$. Figure 13 shows the minimal weight $d$ to reach a final state for all states of the transducer. Using this value, the new transition weights are calculated.
as shown in Figure 14, resulting in the transducer from Figure 15.

Figure 12: Non-minimal transducer before pushing.

Next to applying the pushing operation, the usual minimization algorithm from Section 2.2 can be used. The input and output labels are simply combined to one label. After several refinements of the state partition, the two states 1 and 2, as well as the two states 3 and 4 can be merged. Figure 16 presents the resulting transducer. It is a minimal transducer equivalent to the initial one from Figure 12. Since no more pushing is possible, it is also its unique minimum transducer, of course, except for state names.

Remarks

The complexity of the minimization of subsequential transducers is always as good as that of classical automata minimization: $O(|Q| + |E|)$ for acyclic transducers, and $O(|E| \cdot \log |Q|)$ in general ([Moh97]).

4 Application to Speech Recognition

Subsequential string-to-weight transducers have many applications in speech recognition. As proposed by [MPR96], speech recognition can be decomposed into several stages. In this section, the application of transducers and the determinization and minimization algorithms during these stages is shortly described.

A model of speech recognition tasks above the signal processing level is shown in Figure 17. It begins with $O$ that describes the acoustic observation sequence of an utterance. It is transformed by an acoustic model $A$ into a sequence of context-dependent phones. These are mapped to (context-independent) phones by a context-dependency model $C$. The context-independent phone sequence serves as input for a pronunciation dictionary $D$, mapping phone units to words. Finally, a language model or grammar $G$ analyzes the word sequence in order to produce sentences.

All these stages can be represented by finite-state transducers that output weights or strings and weights. Simple examples are given in [MPR96]. Thus, the domain of speech recognition can be presented by transducer composition:

$$G \circ D \circ C \circ A \circ O$$

Figure 13: Minimal weights to reach final state.

Figure 14: Example of the pushing operation.

Figure 15: Non-minimal transducer after pushing.

Figure 16: Minimal transducer.
Often, the weights are negative logarithms of probabilities. Since only the paths with a high probability are interesting, output weights have to be minimized. Thus, the used operations are addition and \textit{min}, such that the algorithms given for the tropical semiring can be applied.

In practical applications, transducers are of very large size. Consequently, in order to restrict processing time and memory consumption, not all paths should be expanded. To allow for a practical usage of finite-state transducers, approximation methods or other optimizations have to be adopted. Often, \textit{beam pruning} is used. That is, only those values are considered that have less than a certain distance from the minimum value found so far.

Alternatively, the determinization and minimization algorithms for string-to-weight transducers can be applied. Although determinization may increase the number of states and transitions, the contrary is often true in speech processing due to the high degree of redundancy found in many practical applications.

An important advantage of the presented determination algorithm is that it can be applied on-the-fly. Hence, transitions can be expanded on demand, which leads to optimized complexities.

A practical example from [Moh97] illustrates the possibilities to optimize transduction using the two algorithms. A word lattice is an acyclic string-to-weight transducer where each path corresponds to a sentence. The weights are interpreted as negative logarithms of the probability for a sentence, given a sequence of acoustic observations. Figure 19 shows a word lattice obtained in speech recognition for the utterance \textit{Which flights leave Detroit and arrive at Saint Petersburg around nine a.m.?}. It contains 106 states and 359 transitions, and thus, is obviously very complex with respect to the given phrase. However, it contains lots of redundancy, such that applying the determinization and minimization algorithms leads to a much smaller transducer. The result, containing only 25 states and 33 transitions, is shown in Figure 18. Note that both string-to-weight transducers are equivalent. No pruning or other approximation method has been used.

The example presents only one of many applications of finite-state transducers in speech processing. For instance, [PRS94] presents an application to tokenize Chinese text into words and assign pronunciation to them.

5 Conclusions

Transducers are appropriate devices for numerous applications in natural language processing. Generalizing finite-state machines, they require similar properties in order to be usable in practical settings. More precisely, one desires deterministic transducers to avoid backtracking and minimal transducers so as to minimize space complexity.

The type of transducers focused in this work are subsequential string-to-weight transducers. Algorithms for their determinization and minimization have been presented and compared to
their well-known counterparts for finite-state machines.

A number of issues has been left out in this essay. For example, [Moh97] characterizes transducers that admit an equivalent sequential or sub-sequential transducer. Also, we focused on the tropical semiring and omitted the more general case of an arbitrary semiring. For more details, the interested reader is referred to the references.

References


Figure 19: Very complex word lattice containing lots of redundancy (copied from [Moh97]).