Finite-state Machines: Theory and Applications
Unweighted Finite-state Automata

Thomas Hanneforth

Institut für Linguistik
Universität Potsdam

December 10, 2008
Overview

1. Applications of finite-state machines
2. Finite state acceptors and transducers: formal characterization
3. Closure properties and algebra of finite state acceptors
4. Closure properties and algebra of finite state transducers
5. Equivalence transformations on finite state acceptors
6. Equivalence transformations on finite state transducers
7. Decidability properties of unweighted finite-state acceptors and transducers
Applications of Finite-State Machines
Some Applications of Finite-State Machines in Computational Linguistics

- Morphological analysis: lemmatization, word segmentation, segmentation disambiguation
- Spelling correction
- Lexicon representation
- Part-of-Speech-Tagging
- Shallow parsing, Chunking
- Speech recognition, speech synthesis
- Optimality theory
- Language modeling & Statistical language processing
- Statistical machine translation
Outline

2 Finite state acceptors and transducers: formal characterization
   - Finite-state acceptors
   - Finite-state transducers
Finite state acceptors and transducers: formal characterization
Finite-state acceptors

Non-deterministic finite-state acceptor

**Definition (Non-deterministic finite-state acceptor (NFA))**

A *non-deterministic finite-state acceptor* $A$ is a 5-tuple $\langle Q, \Sigma, q_0, F, \delta \rangle$ where

- $Q$ is a non-empty set of *states*
- $\Sigma$ is a non-empty set and called the *alphabet* of $A$
- $q_0 \in Q$, the *start state*
- $F \subseteq Q$, the *set of final states*
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \mapsto 2^Q$, the *transition function*.

**Notes**

- $\delta$ may be a partial function (and usually is)
- Nondeterminism: the transition function $\delta$ maps a state $q$ and an alphabet symbol $a$ to a *set* of successor states.
- A transition may be labeled with $\varepsilon$, the neutral element of concatenation.
Finite-state acceptors

Example (Nondeterministic FSA $A_{lex}$ accepting some animal names)
Finite-state acceptors
Deterministic finite-state acceptor

Definition (Deterministic finite-state acceptor (DFA))

A deterministic finite-state acceptor $A$ is a 5-tuple $\langle Q, \Sigma, q_0, F, \delta \rangle$, where

- $Q$ is a non-empty set of states
- $\Sigma$ is a non-empty set and called the alphabet of $A$
- $q_0 \in Q$, the start state
- $F \subseteq Q$, the set of final states
- $\delta : Q \times \Sigma \mapsto Q$, the transition function.

Notes

- Again, $\delta$ may be a partial function,
- DFSA are by definition $\varepsilon$-free, that is, contain no $\varepsilon$-transitions.
- DFSA and NDFA have the same generative power that is both concepts are equivalent (cf. subset construction).
Finite-state acceptors

Example (Deterministic version of $A_{\text{lex}}$)

- Deterministic acyclic FSA are also called *tries*. Tries are useful for lexicon representation.
Finite-state acceptors

Examples

Example (DFSA $A_{NP}$ accepting English noun phrase patterns)

Example (DFSA $A_{English}$ accepting some English sentences)
Finite-state acceptors
Generalized transition function, language

Definition (Generalized transition function $\delta^*$)

$\delta^*$ is the reflexive and transitive closure of $\delta$

- $\delta^*(q, \varepsilon) = q, \forall q \in Q$
- $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$

Definition (Language of a DFSA $A$)

$L(A) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$
We also say that $L(A)$ is recognized by $A$.

Definition (Regular language)

The language is called regular if there exists some DFA which recognizes it.
Finite-state transducers

Definition

Definition ((Non-deterministic) finite-state transducer (NFST))

A (non-deterministic) finite-state transducer $T$ is a 7-tuple $\langle Q, \Sigma, \Delta, q_0, F, \delta, \sigma \rangle$, where

- $Q$ is a non-empty set of states
- $\Sigma$ is a non-empty set and called the input alphabet of $T$
- $\Delta$ is a non-empty set and called the output alphabet of $T$
- $q_0 \in Q$, the start state
- $F \subseteq Q$, the set of final states
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \mapsto 2^Q$, the transition function.
- $\sigma : Q \times \Sigma \cup \{\varepsilon\} \times Q \mapsto \Delta^*$, the output function.
Finite-state transducers

Alternative definition

- To simplify some definitions, we combine transition and output function to a set of transitions.
- In addition, we restrict the output function to single symbols or $\varepsilon$.

Definition (Normalized finite-state transducer)

A *normalized finite-state transducer* $T$ is a 6-tuple $\langle Q, \Sigma, \Delta, q_0, F, E \rangle$, where

- $Q$ is a non-empty set of states
- $\Sigma$ is a non-empty set, the input alphabet of $T$
- $\Delta$ is a non-empty set, the output alphabet of $T$
- $q_0 \in Q$, the start state
- $F \subseteq Q$, the set of final states
- $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Delta \cup \{\varepsilon\}) \times Q$, the set of transitions.

- Note that every transducer can be transformed into a normalized transducer.
Finite-state transducers

Example ($T_{lex}$ mapping surface forms to morph. features)

Note that *fish* is nondeterministically mapped to \{ NOUN sg, NOUN pl\}
Finite-state transducers

Example (Laughter machine $T_{\text{laugh}}$)

The input string *laugh* is mapped to the infinite set $\{ha^n|n \geq 1\}$
Finite-state transducers

Example (Bracketing machine $T_{bracket}$)

- Every occurrence of $ab$ is enclosed within brackets.
- For example, the input string $cabbabc$ is mapped to $c\{ab\}b\{ab\}c$ by traversing the state sequence 0 0 2 3 4 0 0 2 3 4 0 0
Finite-state transducers

Language

Definition (Language of a FST)

The language $L(T)$ of a FST $T = \langle Q, \Sigma, \Delta, q_0, F, \delta, \sigma \rangle$ is defined in the following way:

$$L(T) = \{ \langle u, v \rangle \mid \delta^*(q_0, u) \cap F \neq \emptyset \land v \in \sigma^*(q_0, u) \}$$

$\delta^*$ is recursively defined:

- $\delta^*(q, \varepsilon) = \{ q \}$ and
- $\delta^*(q, wa) = \bigcup_{q' \in \delta^*(q, w)} \delta(q', a)$

$\sigma^*$ is the generalized output function and defined like this:

- $\sigma^*(q, \varepsilon) = \{ \varepsilon \}$ and
- $\sigma^*(q, wa) = \sigma^*(q, w) \cdot \bigcup_{q' \in \delta^*(q, w)} \sigma(q', a, p)$
Finite-state transducers

Transduction mapping

**Definition (Transduction mapping)**

The transduction mapping $[T] : \Sigma^* \mapsto \Delta^*$ of a FST $T$ is defined as:

$[T](x) = \{ y | \langle x, y \rangle \in L(T) \}$

**Definition (Functional transducer)**

A transducer $T$ is called *functional* if $|\[T](x)| \leq 1$ for all $x \in \Sigma^*$.

**Example**

$T_{\text{bracket}}$ is functional. $T_{\text{lex}}$ is not functional.

**Definition (Ambiguous transducer)**

A transducer $T$ is called *ambiguous*, if $|\[T](x)| > 1$ for some $x \in \Sigma^*$. 
Finite-state transducers

Ambiguous transducers

Definition (Finitely ambiguous transducer)
A transducer $T$ is called \textit{finitely ambiguous}, if $|\mathbb{J}(x)|$ is finite for all $x \in \Sigma^*$. 

Example
$T_{lex}$ is finitely ambiguous.

Definition (Infinitely ambiguous transducer)
A transducer $T$ is called \textit{infinitely ambiguous}, if $|\mathbb{J}(x)|$ is infinite for some $x \in \Sigma^*$. 

Example
$T_{laugh}$ is infinitely ambiguous.
Finite-state transducers
Deterministic finite-state transducer

Definition (Deterministic finite-state transducer (DFST))

A *deterministic finite-state transducer* \( T \) is a 7-tuple \( \langle Q, \Sigma, \Delta, q_0, F, \delta, \sigma \rangle \), where

- \( Q \) is a non-empty set of *states*
- \( \Sigma \) is a non-empty set and called the *input alphabet* of \( T \)
- \( \Delta \) is a non-empty set and called the *output alphabet* of \( T \)
- \( q_0 \in Q \), the *start state*
- \( F \subseteq Q \), the *set of final states*
- \( \delta : Q \times \Sigma \mapsto Q \), the *(deterministic) transition function*.
- \( \sigma : Q \times \Sigma \mapsto \Delta^* \), the *(deterministic) output function*.

Theorem

*Every deterministic transducer is functional.*
Outline

3 Closure properties and algebra of finite state acceptors
- Union
- Concatenation
- Star closure
- Plus closure
- Reversal
- Complementation
- Intersection
- Difference
- Substitution
- Homomorphism
Closure properties and algebra of finite state acceptors
Closure properties and algebra of finite state acceptors

**Definition (Closure of a set)**

Let $S$ be a set and let $f_k$ be a $k$-ary function taking $k$-tuples over $S$ as arguments. We say that $S$ is closed under $f_k$ if for all $a_i \in S$

$$f_k(a_1, a_2, \ldots, a_k) \in S.$$ 

**Note**

Closure properties are important for the modularity based on a specific formalism. They allow to build complex things out of simpler ones by combining them with a number of operations.
Closure properties and algebra of finite state acceptors

Closure properties of regular languages

The set of languages which is recognized by finite-state acceptors (the regular languages) is closed under

- Union
- Concatenation
- Plus and star closure
- Reversal
- Complementation
- Intersection
- Difference
- Homomorphism and substitution
Closure properties and algebra of finite state acceptors

Union

Example (Union of two acceptors)

\[ A_1 \cup A_2 \]
Closure properties and algebra of finite state acceptors

Concatenation

Example (Concatenation of two acceptors)

\[ A_1 \cdot A_2 \]
Closure properties and algebra of finite state acceptors

Star closure

Example (Star (Kleene) closure of an acceptor)

A

\[ A^* \]
Closure properties and algebra of finite state acceptors

Plus closure

Example (Plus closure of an acceptor)

\[ A \]

\[ A^+ \]
Closure properties and algebra of finite state acceptors

Reversal

Definition (Reversal of a string)

The reversal of a string \( w \in \Sigma^* \) – denoted by \( w^R \) – is defined as:

- \( \epsilon^R = \epsilon \)
- \( (a \cdot w)^R = w^R \cdot a, \quad \forall a \in \Sigma \land w \in \Sigma^* \)

Example

\[
\text{obama}^R = bama^R \cdot o = ama^R \cdot bo = ma^R \cdot abo = a^R \cdot mabo = \epsilon \cdot amabo
\]

Definition (Reversal of a string set)

Let \( S \subseteq \Sigma^* \) be a set of strings. The reversal of \( S \) - denoted by \( S^R \) – is defined as: \( S^R = \{ w^R | w \in S \} \).

Theorem

The set of regular languages is closed under reversal.
Closure properties and algebra of finite state acceptors

Reversal

Example (Reversal of an acceptor)
Closure properties and algebra of finite state acceptors

Complementation

Given an alphabet $\Sigma$ and a FSA $A$ you sometimes need a FSA $\overline{A}$ representing all strings $x$ over $\Sigma^*$ which are not in $A$.

Formally: $L(\overline{A}) = \{x \in \Sigma^* | x \neq L(A)\}$ or $L(\overline{A}) = \Sigma^* - L(A)$

The set of regular languages is closed under complementation.
Closure properties and algebra of finite state acceptors

Complementation: algorithm

**Algorithm:**

1. Determinize $A$ and obtain $A'$.
2. Make $A'$ complete by adding a sink state $s$ and adding for each state $q$ and each symbol $a \in \Sigma$ not already used at $q$ a transition $\delta(q, a) = s$.
3. Exchange final and non-final states.
Closure properties and algebra of finite state acceptors

Complementation

Example (Complementation of a finite-state acceptor)

\[ A \]

\[ (\Sigma = \{a, c, r, t\}) \]
Closure properties and algebra of finite state acceptors

Complementation

Example (Why complementation works)

Consider a trie for \( W = \{\text{cat, camel, dog, frog}\} \).

![Trie Diagram]

Definition (Definition of a trie)

Let \( W \) be a finite set of words over \( \Sigma \). Let \( \text{Pref}(W) \) the set of all prefixes of \( W \). Define a DFA \( A = \langle \text{Pref}(W), \Sigma, \varepsilon, W, \delta \rangle \) with

\[
\forall a \in \Sigma, \; x, xa \in \text{Pref}(W): \; \delta(x, a) = xa.
\]
Closure properties and algebra of finite state acceptors

Complementation

Example (A trie for $W = \{\text{cat, camel, dog, frog}\}$)

States are labeled with prefixes of $W$. 

![Diagram of a trie for the set $W$ containing the strings cat, camel, dog, and frog. The diagram shows states labeled with prefixes of $W$, with transitions indicated by arrows.]
Closure properties and algebra of finite state acceptors

Complementation

- In a trie – a special acyclic DFA – each state corresponds to a single prefix of a word in \( W \).
- In a general DFA \( A \), each state \( q \) corresponds to a set of prefixes of the words in \( L(A) \), the left language of \( q \).

**Definition (Left language)**

The left language of a state \( q \) – symbolically \( \overleftarrow{L}(q) \) – is defined as:

\[
\overleftarrow{L}(q) = \{ w \in \Sigma^* \mid \delta^*(q_0, w) = q \}
\]

**Example (Left language)**

\[
\overleftarrow{L}(1) = \{ a(ba)^n \mid n \geq 0 \}
\]
Closure properties and algebra of finite state acceptors

Complementation: Why can we only complementize DFAs?

Example (’Complementation’ of an NFA)

The ”complementized” NFA still accepts $ab$. 
Complementation (negation) is important for the inherent robustness of methods based on finite-state automata.

A NLP system is called robust if there does not exist an input string for which it fails. That means: A robust NLP system accepts $\Sigma^*$.

This is immediately related to negation: if there is some input string $w$ which is not accepted by some DFA $A$ (say, for example, a DFA representing some NP grammar about stock indices), one could use $\overline{A}$ to accept $w$: $L(A) \cup L(\overline{A}) = \Sigma^*$.

This property does not carry over to context-free languages: they are not closed under complementation.

Context-sensitive languages are – perhaps surprisingly – again closed under complementation. But their recognition problem is notoriously difficult.
Closure properties and algebra of finite state acceptors

Complementation: the sting in the tail: complexity

**Theorem**

Consider an NFA $A$ with state set $Q$. The state complexity of an equivalent DFA $A'$ can be in the worst case in $O(\lvert \Sigma \rvert 2^{|Q|})$.

**Example (Worst case of determinization)**

Consider the regular language $L = \Sigma^* a (a|b)^k$ for some $k$ (with $\Sigma = \{a, b\}$). While an NFA for $L$ has $k + 2$ states, the equivalent DFA has $2^{k+1}$ states.

$k = 2$
Closure properties and algebra of finite state acceptors

Intersection

- If we know that FSAs are closed under complementation and union then we also know that they are closed under intersection.
- Why? By DeMorgan!
  \[ A \cap B \equiv \overline{A} \cup \overline{B} \]
- But this approach is very complex, since it requires three complementation operations which in turn require determinization.
- There is a more direct method: we let pair of states of the original FSAs be states of the intersection FSA.
Closure properties and algebra of finite state acceptors

Intersection: Example

Example (Intersection with the product state construction)

\[
\begin{align*}
A_1 & \quad 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{a} 3 \xrightarrow{c} 4 \\
A_2 & \quad 0 \xrightarrow{a} 1 \quad 2 \xrightarrow{b} 3 \xrightarrow{a} 2 \xrightarrow{c} 3 \\
A_1 \cap A_2 & \quad <0,0> \xrightarrow{a} <1,1> \xrightarrow{b} <2,2> \xrightarrow{a} <3,1> \xrightarrow{c} <4,3>
\end{align*}
\]
Closure properties and algebra of finite state acceptors

Intersection: formal definition

Definition (Intersection of two finite-state acceptors)

Let $A_1 = \langle Q_1, \Sigma_1, q_{01}, F_1, \delta_1 \rangle$ and $A_2 = \langle Q_2, \Sigma_2, q_{02}, F_2, \delta_2 \rangle$ be two FSAs. $A_1 \cap A_2$, the intersection of $A_1$ and $A_2$ is an acceptor:

$$A = \langle Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \langle q_{01}, q_{02} \rangle, F_1 \times F_2, \delta \rangle$$

where $\langle p', q' \rangle \in \delta(\langle p, q \rangle, a)$ if $p' \in \delta_1(p, a)$ and $q' \in \delta_2(q, a)$ for all $a \in \Sigma_1 \cap \Sigma_2$.

- This mathematical approach generates in the worst as in the best case a FSA with $|Q_1||Q_2|$ states.
- But a lot of these states may not contribute to the language.
Closure properties and algebra of finite state acceptors

Intersection: useless states

Definition (Inaccessible and non-coaccessible states)

Let $A$ be a finite-state automaton (acceptor or transducer) with start state $q_0$. A state $q$ in $A$ is called \textit{inaccessible} if there is no path in $A$ from $q_0$ to $q$. A state $q$ in $A$ is called \textit{non-coaccessible} if there is no path in $A$ from $q$ to a final state of $A$. A state is called \textit{useless} if it is inaccessible or non-coaccessible. A finite-state automaton $A$ is called \textit{trim} or \textit{connected} if it has no useless states.
Closure properties and algebra of finite state acceptors

Intersection: removal of useless states

**Algorithm** \( \text{connect}(A) \)

**Require:** FSM \( A \) with start state \( q_0 \), state set \( Q \) and final state set \( F \)

**Ensure:** \( A \) without useless states

1. Perform a depth-first search starting at \( q_0 \) and mark each visited state
2. Delete each unmarked state \( q \) and all its ingoing and outgoing transitions
3. Reverse \( A \)
4. Unmark all states in \( Q \)
5. Perform a depth-first search starting at all states \( q \in F \) and mark each visited state
6. Delete each unmarked state \( q \) and all its ingoing and outgoing transitions
7. Reverse \( A \)

**Complexity of** \( \text{connect}(A) \)

If \( A \) has \( |Q| \) number of states and \( |E| \) number of transitions, the complexity of \( \text{connect}(A) \) is in \( \mathcal{O}(|Q| + |E|) \).
Closure properties and algebra of finite state acceptors

Intersection: algorithm

Require: FSAs $A_1 = \langle Q_1, \Sigma_1, q_{01}, F_1, \delta_1 \rangle$ and $A_2 = \langle Q_2, \Sigma_2, q_{02}, F_2, \delta_2 \rangle$

Ensure: $A = A_1 \cap A_2$

1: $F := Q := \emptyset$
2: $ENQUEUE(S, \langle q_{01}, q_{02} \rangle)$
3: \textbf{while} $S \neq \emptyset$ \textbf{do}
4: \hspace{1em} $\langle q_1, q_2 \rangle := DEQUEUE(S)$
5: \hspace{2em} \textbf{for all} $a \in \Sigma_1 \cap \Sigma_2$ \textbf{do}
6: \hspace{3em} \textbf{if} $q_1' \in \delta_1(q_1, a) \land q_2' \in \delta_2(q_2, a)$ \textbf{then}
7: \hspace{4em} $\delta(\langle q_1, q_2 \rangle, a) := \delta(\langle q_1, q_2 \rangle, a) \cup \{\langle q_1', q_2' \rangle\}$
8: \hspace{3em} \textbf{if} $\langle q_1', q_2' \rangle \notin Q$ \textbf{then}
9: \hspace{4em} $Q := Q \cup \{\langle q_1', q_2' \rangle\}$
10: \hspace{3em} \textbf{if} $q_1' \in F_1 \land q_2' \in F_2$ \textbf{then}
11: \hspace{4em} $F := F \cup \{\langle q_1', q_2' \rangle\}$
12: \hspace{3em} \textbf{end if}
13: \hspace{2em} $ENQUEUE(S, \langle q_1', q_2' \rangle)$
14: \hspace{1em} \textbf{end if}
15: \hspace{1em} \textbf{end if}
16: \hspace{1em} \textbf{end for}
17: \textbf{end while}
18: $CONNECT(A)$
19: \textbf{return} $A$
Closure under intersection means that we can develop constraints independently of each other and then enforce their validity simultaneously by intersecting them.

A lot of finite-state based NLP is based on intersection: (Two-level-) Morphology, Constraint based grammar, pattern matching etc.
Definition (Difference)
Let $A_1$ and $A_2$ two FSAs. The difference $A_1 - A_2$ is defined as:

$$A_1 - A_2 \equiv A_1 \cap \overline{A_2}$$
Closure properties and algebra of finite state acceptors

Difference

Example (Difference)

\[ A_1 - A_2 \]
Closure properties and algebra of finite state acceptors

Substitution

**Definition (Substitution)**

A *substitution* is a mapping $s : \Sigma \mapsto 2^{\Delta^*}$ for two alphabets $\Sigma$ and $\Delta$. $s$ is generalized to $s^* : \Sigma^* \mapsto 2^{\Delta^*}$ by:

- $s^*(\varepsilon) = \varepsilon$
- $s^*(xa) = s^*(x)s(a)$

**Theorem (Closure under substitution)**

The set of regular languages is closed under substitution with regular languages.

**Note**

A lot of finite-state based NLP is based on closure under substitution.
Closure properties and algebra of finite state acceptors

Substitution

Example (Substitution in computational morphology)

A morphology rule as a FSA $A$

Result of the substitution $A\{STEM = A_1, NINF = A_2\}$
Definition (Homomorphism)

A homomorphism is a mapping $h : \Sigma \rightarrow \Delta^*$ for two alphabets $\Sigma$ and $\Delta$.

Definition (Inverse homomorphism)

Given a homomorphism $h$, the inverse homomorphic image $h^{-1}$ of a language $L$ is defined as: $h^{-1}(L) = \{w \mid h(w) \in L\}$

Theorem (Closure under homomorphism)

The set of regular languages is closed under homomorphism and inverse homomorphism.
Outline

4 Closure properties and algebra of finite state transducers
- Projection
- Composition
- Cross product
- Inversion
- Intersection
Closure properties and algebra of finite state transducers
Closure properties and algebra of finite state transducers

The set of finite state transducers is closed under

- Union
- Concatenation
- Closure
- Reversal
- Projection (note that this leads to FSAs)
- Composition
- Inversion

Finite state transducers are **not** closed under

- Complementation
- Intersection (but acyclic and $\varepsilon$-free transducers are)
- Difference
Closure properties and algebra of finite state transducers

Projection

**Definition (First and second projection)**

Let $T = \langle Q, \Sigma, \Delta, q_0, F, S \rangle$ be a transducer. The *first projection* of $T$ – symbolically $\pi_1(T)$ – is the FSA $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ where

$$\forall a \in \Sigma \cup \{\varepsilon\}, \quad \delta(p, a) = \{ q \mid \exists b \in \Delta : \langle p, a, b, q \rangle \in S \}$$

The *second projection* of $T$ – symbolically $\pi_2(T)$ – is the FSA $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ where

$$\forall b \in \Delta \cup \{\varepsilon\}, \quad \delta(p, b) = \{ q \mid \exists a \in \Sigma : \langle p, a, b, q \rangle \in S \}$$
Closure properties and algebra of finite state transducers

Projection

Example (Projection)

Transducer $T$

$\pi_1(T')$

$\pi_2(T')$
Closure properties and algebra of finite state transducers

Composition

Composing a transducer $T_1$ with a transducer $T_2$ (formally $T_1 \circ T_2$) means: take some input $u$ for $T_1$, collect the output $v$ of $T_1$, feed it as input into $T_2$ and collect the output $w$ of $T_2$.

Definition (Composition relation)

Let $T_1 = \langle Q_1, \Sigma_1, \Delta_1, q_{01}, F_1, S_1 \rangle$ and $T_2 = \langle Q_2, \Sigma_2, \Delta_2, q_{02}, F_2, S_2 \rangle$ be transducers. $L(T_1 \circ T_2) = \{(u, w) \in \Sigma_1^* \times \Delta_2^* | \exists v \in \Delta_1^* \cap \Sigma_2^*: \langle u, v \rangle \in L(T_1) \land \langle v, w \rangle \in L(T_2)\}$
Closure properties and algebra of finite state transducers

Composition

**Definition (ε-free composition)**

Let $T_1 = \langle Q_1, \Sigma_1, \Delta_1, q_{01}, F_1, E_1 \rangle$ and $T_2 = \langle Q_2, \Sigma_2, \Delta_2, q_{02}, F_2, E_2 \rangle$ be two normalized, ε-free FSTs. $T_1 \circ T_2$, the composition of $T_1$ and $T_2$, is the transducer $T = \langle Q_1 \times Q_2, \Sigma_1, \Delta_2, \langle q_{01}, q_{02} \rangle, F_1 \times F_2, E \rangle$ where

$$E = \{ \langle \langle p, q \rangle, a, b, \langle p', q' \rangle \rangle \mid \exists c \in \Delta_1 \cap \Sigma_2 : \langle p, a, c, p' \rangle \in E_1 \wedge \langle q, c, b, q' \rangle \in E_2 \}$$

**Properties of composition**

- The composition operation is *not commutative*, that is, in general:
  $$T_1 \circ T_2 \neq T_2 \circ T_1$$
- The composition operation is *associative*, that is:
  $$T_1 \circ T_2 \circ T_3 = (T_1 \circ T_2) \circ T_3 = T_1 \circ (T_2 \circ T_3)$$
How does composition work?

Whenever $T_1$ contains a transition: $T$ will contain a transition:

Whenever $T_2$ contains a transition:

$T$ will contain a transition:
Closure properties and algebra of finite state transducers

Composition

Example (Composition)

FST repeatedly mapping words to their categories

FST mapping NP-patterns to NP category
Definition (Identity transducer)

Let $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a FSA.

The identity transducer $ID(A)$ is defined by $\langle Q, \Sigma, \Sigma, q_0, F, E \rangle$ where

$$E = \{ \langle p, a, a, q \rangle \mid \exists p, q \in Q, a \in \Sigma \cup \{ \varepsilon \} : q \in \delta(p, a) \}$$

Example (Identity transducer)

![Diagram of an identity transducer](image)

Definition (Application)

The application of a FST $T$ to a FSA $A$ – symbolically $T[A]$ – is defined as

$$T[A] \equiv \pi_2(ID(A) \circ T)$$
Closure properties and algebra of finite state transducers

Application

Example

FSA $A$

FSA $ID(A)$

$ID(A) \circ T$

FST $T$

$T[A] = \pi_2(ID(A) \circ T)$
Composition can be considered as a generalization of intersection. The intersection of two FSAs $A_1$ and $A_2$ can be defined as follows:

$$A_1 \cap A_2 = \pi_1(ID(A_1) \circ ID(A_2))$$

So, intersecting two FSAs is done by composing their identity transducers and afterwards projecting one of the tapes. Composing two transducers $X$ and $Y$ means synchronizing (intersecting) their inner tapes and then combining the outer tapes:
Closure properties and algebra of finite state transducers

Composition: handling \(\varepsilon\)-transitions

It is possible to generalize the composition definition to transducers with \(\varepsilon\)-transitions:

**Definition (Transducer composition)**

Let \(T_1 = \langle Q_1, \Sigma_1, \Delta_1, q_{01}, F_1, E_1 \rangle\) and \(T_2 = \langle Q_2, \Sigma_2, \Delta_2, q_{02}, F_2, E_2 \rangle\) be two normalized FSTs.

\(T_1 \circ T_2\), the composition of \(T_1\) and \(T_2\), is the transducer

\[
T = \langle Q_1 \times Q_2, \Sigma_1, \Delta_2, \langle q_{01}, q_{02} \rangle, F_1 \times F_2, E \cup E_\varepsilon \cup E_{i,\varepsilon} \cup E_{o,\varepsilon} \rangle
\]

where

\[E = \{\langle\langle p, q \rangle, a, b, \langle p', q' \rangle\rangle \mid \exists c \in \Delta_1 \cap \Sigma_2 : \langle p, a, c, p' \rangle \in E_1 \land \langle q, c, b, q' \rangle \in E_2\}\]

\[E_\varepsilon = \{\langle\langle p, q \rangle, a, b, \langle p', q' \rangle\rangle \mid \langle p, a, \varepsilon, p' \rangle \in E_1 \land \langle q, \varepsilon, b, q' \rangle \in E_2\}\]

\[E_{i,\varepsilon} = \{\langle\langle p, q \rangle, \varepsilon, a, \langle p, q' \rangle\rangle \mid \langle q, \varepsilon, a, q' \rangle \in E_2 \land p \in Q_1\}\]

\[E_{o,\varepsilon} = \{\langle\langle p, q \rangle, a, \varepsilon, \langle p', q \rangle\rangle \mid \langle p, a, \varepsilon, p' \rangle \in E_1 \land q \in Q_2\}\]
Closure properties and algebra of finite state transducers

Composition: handling $\varepsilon$-transitions

There are four different ways, how $\varepsilon$ and alphabet symbols on the second tape of $T_1$ and the first tape of $T_2$ can interact:

1. $T_1$ contains a $a : c$-transition and $T_2$ contains a $c : b$-transition: this is handled in the same way as in the $\varepsilon$-free case.

2. $T_1$ contains a $a : \varepsilon$-transition and $T_2$ contains a $\varepsilon : b$-transition $\rightarrow$ $T$ contains a $a : b$-transition. That is: $\varepsilon$ is treated as a regular symbol.

3. $T_1$ “stays” in the same state, $T_2$ moves on:

4. $T_1$ moves on, $T_2$ “stays” in the same state:
Closure properties and algebra of WFSA

Composition

Example (Composition of two unweighted FSTs)

\[ T_1 \circ T_2 \]
Composition is a very important operation for building processing or filtering cascades, for example in robust parsing and morphological analysis.

Since composition is not commutative, the order of a transducer cascade $C = T_1 \circ T_2 \circ \ldots \circ T_k$ matters.

This may lead to problems related to feeding, counter-feeding, bleeding and counter-bleeding.

Since the composition operation is associative, the order in which the compositions in $C$ are computed does not matter. This entails some freedom degrees for implementing such cascades.

Note, that the state complexity of $T_1 \circ T_2 \circ \ldots \circ T_k$ is $|Q_1||Q_2|\ldots|Q_k|$ in the worst case.
Closure properties and algebra of finite state transducers

Cross product

**Definition ((Cartesian) Product)**
Given two sets $S_1$ and $S_2$, the Cartesian product $S_1 \times S_2$ is defined as:

$$S_1 \times S_2 = \{ \langle x, y \rangle \mid x \in S_1 \land y \in S_2 \}$$

**Theorem (Product of regular sets)**

Let $A_1 = \langle Q_1, \Sigma_1, q_{01}, F_1, \delta_1 \rangle$ and $A_2 = \langle Q_2, \Sigma_2, q_{02}, F_2, \delta_2 \rangle$ be two finite-state acceptors. Then $L(A_1) \times L(A_2)$ is representable by a finite-state transducer $A_1 \times A_2$.

**Proof.**

$$A_1 \times A_2 \equiv ID(A_1) \circ T_{\Sigma_1^*\rightarrow \varepsilon} \circ T_{\varepsilon\rightarrow \Sigma_2^*} \circ ID(A_2)$$

**Note**

Cross product is the core of all replacement operations.
Closure properties and algebra of finite state transducers

Cross product

Example (Cross product)

\[ A_1 \times A_2 \]
Inversion

**Definition (Inversion)**

Let $T = \langle Q, \Sigma, \Delta, q_0, F, E \rangle$ be a transducer. The *inversion* of $T$ – symbolically $T^{-1}$ – is the FST $T^{-1} = \langle Q, \Delta, \Sigma, q_0, F, E^{-1} \rangle$ where $E^{-1} = \{ \langle p, b, a, q \rangle | \langle p, a, b, q \rangle \in E \}$.

**Note**

Thus, inversion simply exchanges input- and output “tapes” of a transducer.
Closure properties and algebra of finite state transducers

Inversion

Example (Morphological analysis vs. generation)

FST $T_{Morph}$ mapping words to morphological categories

FST $T_{Morph}^{-1}$ mapping morphological categories to words
Closure properties and algebra of finite state transducers

Why are FSTs not closed under intersection?

**Example**

The intersection of $T_{a^n \rightarrow b^n c^*}$ and $T_{a^n \rightarrow b^* c^n}$ would result in the relation $R = \{ \langle a^n, b^n c^n \rangle \mid n \geq 0 \}$ which is not regular and thus not representable by a finite-state transducer.

**Note**

This has consequences for creating applications based on finite-state transducers. They cannot be based on the intersection of constraints represented as transducers.
Closure properties and algebra of finite state transducers

Why are FSTs not closed under intersection?

- Intuitively, the existence of $\varepsilon$ within loops leading to infinite ambiguity is the reason why FSTs are not closed under intersection.
- Thus, $\varepsilon$-free FSTs – also called equal-length transducers – are closed under intersection.
- The same is true for acyclic FSTs, where we have some freedom where to realize the $\varepsilon$-transitions.
- By DeMorgan, non-closure under intersection leads to non-closure under complementation.
Outline

5 Equivalence transformations on finite-state acceptors
   • $\varepsilon$-Removal
   • Determinization
   • Minimization
Equivalence transformations on finite-state acceptors
Equivalence transformations on finite-state acceptors

Equivalence transformations

- *Equivalence transformations* are operations on automata which change the topology of an automaton without changing its language.
- They usually serve optimization purposes, that is, they create smaller and/or faster automata.

Finite-state acceptors admit the following equivalence transformations:

- $\varepsilon$-Removal
- Determinization
- Minimization
Equivalence transformations on finite-state acceptors

$\varepsilon$-Removal

**Definition ($\leadsto$)**
Let $p$ and $q$ be states in $Q$ and let $w$ be a string in $\Sigma^*$. Let $\leadsto^w$ be a relation $Q \times Q$, such that $\langle p, q \rangle \in \leadsto^w$ if there is a path labeled with $w$ from $p$ to $q$.

**Definition ($\varepsilon$-closure)**
Given a NFA $A = \langle Q, \Sigma, q_0, F, \delta \rangle$, $\varepsilon$-closure$(q) = \{q\} \cup \{p \in Q \mid q \leadsto^\varepsilon p\}$.

**Definition ($\varepsilon$-free FSA)**
Let $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a FSA. Define $A'$, the equivalent $\varepsilon$-free FSA with $L(A') = L(A)$, as $A' = \langle Q, \Sigma, q_0, F', \delta' \rangle$ where:

- $\delta'(q, a) = \varepsilon$-closure($\bigcup_{q' \in \varepsilon$-closure$(q)} \delta(q', a)$), $\forall q \in Q, a \in \Sigma$
- $F' = F \cup \{q_0\}$, if $\varepsilon$-closure$(q_0) \cap F \neq \emptyset$, else $F' = F$. 
Equivalence transformations on finite-state acceptors

$\varepsilon$-Removal

Example
Equivalence transformations on finite-state acceptors

Determinization

Definition (Subset construction)

Let $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a FSA. Define $A'$, the equivalent DFA with $L(A') = L(A)$ as $A' = \langle 2^Q, \Sigma, \{q_0\}, F', \delta' \rangle$ with:

- $F' = \{ S \subseteq Q \mid S \cap F \neq \emptyset \}$
- $\delta'(S, a) = \bigcup_{q \in S} \delta(q, a)$, $\forall a \in \Sigma$, $\forall S \subseteq Q$

Note

- The complexity of an algorithm which implements this in a naive way is exponential.
- In the normal case, most of the subset states in the DFA are not accessible / coaccessible.
- A better algorithm based on a state queue avoids inaccessible states.
- But this doesn’t change the complexity in the worst case.
Equivalence transformations on finite-state acceptors

Determinization

Example
Equivalence transformations on finite-state acceptors

Minimization

**Definition (Minimal DFA)**

Let $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a DFA.

$L$ is minimal if $\forall A' = \langle Q', \Sigma', q'_0, F', \delta' \rangle : L(A') = L(A) \Rightarrow |Q| \leq |Q'|$

**Definition (Right language)**

Let $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a DFA.

The right language of a state $q \in Q$ – symbolically $\overrightarrow{L}(q)$ – is defined as:

$$\overrightarrow{L}(q) = \{ w \in \Sigma^* | \delta^*(q, w) \in F \}$$
Equivalence transformations on finite-state acceptors

Minimization

Definition (Equivalent states)

Two states $p$ and $q$ are called equivalent if $\overrightarrow{L}(p) = \overrightarrow{L}(q)$.

This holistic definition based on right languages can be turned into a recursive definition of equivalence of states:

Definition (State equivalence $\equiv$)

Let $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a DFA.

Two states $p$ and $q$ are called equivalent – symbolically $p \equiv q$ –, if:

$$p \equiv q \text{ if } p \in F \iff q \in F \land \forall a \in \Sigma : \delta(p, a) \equiv \delta(q, a).$$
Equivalence transformations on finite-state acceptors

Minimization

Based on state equivalence, we come up with a definition of a minimal DFA:

**Theorem (Minimal DFA I)**

Let $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a DFA. $A$ is minimal iff

$$\forall p, q \in Q : p \neq q \implies \overrightarrow{L}(p) \neq \overrightarrow{L}(q).$$

By substituting the recursive definition of state equivalence into the last theorem, we arrive at:

**Theorem (Minimal DFA II)**

Let $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a DFA. $A$ is minimal iff

$$\forall p, q \in Q : p \neq q \implies p \in F \iff q \in F \lor \exists a \in \Sigma : \delta(p, a) \neq \delta(q, a).$$
Theorem (Myhill-Nerode)

The following propositions are equivalent:

1. \( L \) is recognized by a DFA \( A_L = \langle Q, \Sigma, q_0, F, \delta \rangle \).
2. \( L \) is the union of some equivalence classes of a right invariant equivalence relation \( R \) with finite index.
3. \( R_L (x \ R_L y \ \iff \ \forall z \in \Sigma^*: xz \in L \iff yz \in L) \) is of finite index.

Notes

- The Myhill-Nerode theorem links states with subsets of \( \Sigma^* \). It is central to the theorem that the number of states in a DFA is finite.
- The Myhill-Nerode theorem assumes complete DFAs, that is, the corresponding transition function \( \delta \) is total.
Equivalence transformations on finite-state acceptors

Minimization: Myhill-Nerode theorem

**Definition (Equivalence relation, equivalence class)**

Let $S$ be a set and $E \subseteq S \times S$ a binary relation. $E$ is called a *equivalence relation* if $E$ is reflexive, symmetric and transitive.

If $E$ is a equivalence relation, we call $[x]_E = \{y \mid x E y\}$ the *equivalence class* of $x$ wrt $E$.

**Properties of equivalence relations**

1. $x \in [x]_E, \forall x \in S$
2. $[x]_E = [y]_E \lor [x]_E \cap [y]_E = \emptyset, \forall x, y \in S$
3. $\bigcup_{x \in S} [x]_E = S$

**Definition (Index of a equivalence relation)**

Let $E$ be a equivalence relation. The *index* $I_E$ of $E$ is the number of $E$’s equivalence classes. $E$ is of *finite index* if $I_E$ is finite.
Equivalence transformations on finite-state acceptors

Minimization: Myhill-Nerode theorem

**Definition (Right-invariant equivalence relation)**

Let $R$ be an equivalence relation over $\Sigma^*$. $R$ is called *right-invariant* (with respect to concatenation) if

$$\forall x, y, z \in \Sigma^* : x R y \Rightarrow xz R yz$$

**Definition (Left language)**

Let $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a DFA. Define the left language $\overleftarrow{L}(p)$ of a state $p \in Q$ as

$$\overleftarrow{L}(p) = \{ w \in \Sigma^* | \delta(q_0, w) = p \}.$$
Equivalence transformations on finite-state acceptors

Minimization: Myhill-Nerode theorem

Myhill-Nerode theorem.

We prove the theorem by chaining \( 1 \Rightarrow 2 \), \( 2 \Rightarrow 3 \) and \( 3 \Rightarrow 1 \).

\( 1 \Rightarrow 2 \).

Let \( A \) be a DFA recognizing \( L \).

Define \( R_A \) as \( x R_a y \) if \( \delta(q_0, x) = \delta(q_0, y), \forall x, y \in \Sigma^* \).

Subproof: \( R_A \) is right-invariant equivalence relation of finite index (the index of \( R_A \) is \( |Q| \)). The equivalence classes of \( R_A \) are the left languages \( \overleftarrow{L}(p), \forall p \in Q \). \( \bigcup_{q \in F} \overleftarrow{L}(q) = L \).

Example

\[ \overleftarrow{L}(1) = \{ \varepsilon, a(ba)^*b \}, \overleftarrow{L}(2) = \{ a(ba)^* \}, \overleftarrow{L}(3) = \{ a(ba)^*c(d)^* \} \]

\( L = \overleftarrow{L}(2) \cup \overleftarrow{L}(3) \)
Equivalence transformations on finite-state acceptors
Minimization: Myhill-Nerode theorem

Myhill-Nerode theorem (continued).

2 $\Rightarrow$ 3.

$R = R_A$ is an \textit{refinement} of $R_L$, that is, every equivalence class of $R_A$ is contained in some equivalence class of $R_L$.

1. Assume that $x \ R_A \ y$.
2. Since $R_A$ is right-invariant, $xz \ R_A \ yz$, for all $z \in \Sigma^*$.
3. Thus $xz \in L$ if and only if $yz \in L$.
4. Thus $xz \ R_L \ yz$ and the equivalence class of $x \ \text{wrt} \ R_A$ is contained in the equivalence class of $x \ \text{wrt} \ R_L$.
5. Since the index of $R_A$ is finite (at most equal to $|Q|$) and the index of $R_L$ is less or equal to the index of $R_A$, we conclude that $R_L$ is of finite index.
Equivalence transformations on finite-state acceptors

Minimization: Myhill-Nerode theorem

Example ($R_A$ is a refinement of $R_L$)

<table>
<thead>
<tr>
<th>State</th>
<th>Corresponding equiv. class of $R_A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>e</td>
</tr>
<tr>
<td>9</td>
<td>frie</td>
</tr>
<tr>
<td>15</td>
<td>dog</td>
</tr>
<tr>
<td>17</td>
<td>doll</td>
</tr>
<tr>
<td>19</td>
<td>dollar</td>
</tr>
<tr>
<td>22</td>
<td>coll</td>
</tr>
<tr>
<td>24</td>
<td>collar</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Corresponding equiv. class of $R_L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>dog, collar, dollar, end, friend, frog</td>
</tr>
<tr>
<td>8</td>
<td>e, frie</td>
</tr>
<tr>
<td>12</td>
<td>doll</td>
</tr>
<tr>
<td>16</td>
<td>coll</td>
</tr>
</tbody>
</table>

Thomas Hanneforth (Universität Potsdam)  Finite-state Machines: Theory and Applications  December 10, 2008  91 / 99
Equivalence transformations on finite-state acceptors

Minimization: Myhill-Nerode theorem

Myhill-Nerode theorem (continued).

3 ⇒ 1.

Given $R_L$, construct a new FSA $A' = \langle Q', \Sigma, q_0', F', \delta' \rangle$ as follows:

1. $Q' = \{[x] \mid [x] \text{ is a equivalence class of } \Sigma^* \text{ under } R_L \}$
2. $q_0' = [\varepsilon]$
3. $\delta' : Q' \times \Sigma \mapsto Q' : \delta'([x], a) = [xa], \forall [x] \in Q' \land a \in \Sigma$
4. $F' = \{[x] \mid x \in L\}$

Example

$\delta'([e], n) = [en]$
$\delta'([en], d) = [end]$
$\delta'(\{frie, e\}, n) = \{frien, en\}$
$\delta'(\{frien, en\}, d) = \{friend, end\}$

*a Inspired by CAKE: “Friend is a four-letter word”*
Equivalence transformations on finite-state acceptors
Minimization: algorithms

Approaches

1. **Union-Find-based**: Find all states $p$ and $q$ with $\overrightarrow{L}(p) = \overrightarrow{L}(q)$ and merge them.

2. **Partition-based**: Starting at sets of non-equivalent states, partition these sets further until each set contains only equivalent states.
Outline

6 Equivalence transformations on finite-state transducers
Equivalence transformations on finite-state transducers
Decidability properties of unweighted finite-state acceptors and transducers
Decidability properties of unweighted finite-state acceptors and transducers
Decidability properties of unweighted finite-state acceptors and transducers

Given two finite-state acceptors $A$ and $A'$, the following properties are decidable:

- $L(A) = \emptyset$
- $L(A) = \Sigma^*$
- $L(A) = L(A')$
- $L(A) \subseteq L(A')$

Given two finite-state transducers $T$ and $T'$, the following properties are decidable:

- $T$ is functional

Given two finite-state transducers $T$ and $T'$, the following properties are undecidable:

- $L(T) = L(T')$
Version history

- 16.10.08: version 0.1 (initial version)
- 20.10.08: version 0.2 (some error corrections, added definition of composition of FSTs with $\varepsilon$-transitions)
- 09.11.08: version 0.3 (added example for $\varepsilon$-composition, enhanced example for cross product, added subtitles)
- 07.12.08: version 0.4 (completed minimization section)