Chapter 1

String Acceptors

1.1 Deterministic Finite-state String Acceptors

1.1.1 Orientation

This section is about deterministic finite-state acceptors for strings. The term *finite-state* means that the memory is bounded by a constant, no matter the size of the input to the machine. The term *deterministic* means there is single course of action the machine follows to compute the output from some input. As we will see later, *non-deterministic machines* can be thought of as pursuing multiple computations simultaneously. The term *acceptor* is synonymous with *recognizer*. It means that this machine solves *membership problems*: given a set of objects X and input object x, does x belong to X? The term *string* means we are considering the membership problem over stringsets. So X is a set of strings (so $X \subseteq \Sigma^*$) and the input x is a string.

1.1.2 Definitions

Definition 1. *A* deterministic finite-state acceptor (DFA) *is a tuple* $(Q, \Sigma, q_0, F, \delta)$ *where*

- Q *is a finite set of states;*
- Σ *is a finite set of symbols (*the alphabet*);*
- $q_0 \in Q$ *is the* initial *state;*
- $F \subseteq Q$ *is a set of* accepting *(*final*) states; and*
- δ *is a function with domain* $Q \times \Sigma$ *and co-domain* Q*. It is called the* transition function.

We extend the domain of the transition function to $Q \times \Sigma^*$ as follows. In these notes, the empty string is denoted with λ.

$$\begin{aligned} \delta^*(q, \lambda) &= q \\ \delta^*(q, aw) &= \delta^*((\delta(q, a), w) \end{aligned} \tag{1.1}$$

Consider some DFA $A = (Q, \Sigma, q_0, F, \delta)$ and string $w \in \Sigma^*$. If $\delta^*(q_0, w) \in F$ then we say A *accepts/recognizes/generates* w. Otherwise A *rejects* w.

Definition 2. *The stringset recognized by A is $L(A) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$.*

The use of the 'L' denotes "Language" as stringsets are traditionally referred to as *formal languages.*

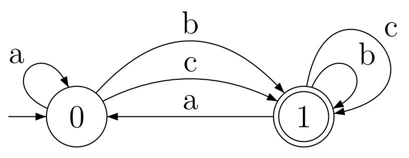
Definition 3. *A stringset is* regular *if there is a DFA that recognizes it.*

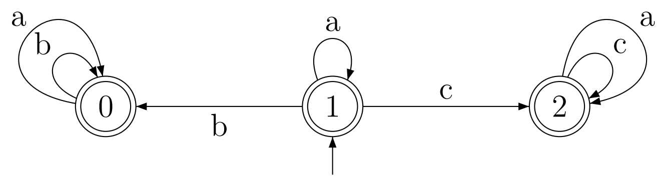
1.1.3 Exercises

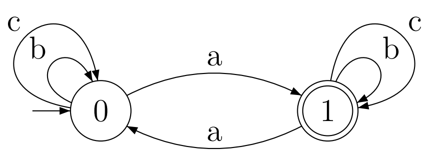
Exercise 1. This exercise is about designing DFA. Let $\Sigma = \{a, b, c\}$. Write DFA which express the following generalizations on word well-formedness.

1. All words begin with a consonant, end with a vowel, and alternate consonants and vowels.
2. Words do not contain *aaa* as a substring.
3. If a word begins with *a*, it must end with *c*.
4. Words must contain two *b*s.
5. All words have an even number of vowels.

Exercise 2. This exercise is about reading and interpreting DFA. Provide generalizations in English prose which accurately describe the stringset these DFA describe.

1. 

2. 

3. 

4. Write the DFA in #1-3 in mathematical notation. So what is Q, Σ, q_0, F, and δ?

1.2 Properties of DFA

Note that for a DFA A, its transition function δ may be partial. That is, there may be some $q \in Q, a \in \Sigma$ such that $\delta(q, a)$ is undefined. If δ is a partial function, δ^* will be also. It is assumed that if $\delta^*(q_0, w)$ is undefined, then A rejects w.

We can always make δ total by adding one more state to Q. To see how, call this new state $\diamondsuit$. Then for each $(q, a) \in Q \times \Sigma$ such that $\delta(q, a)$ is undefined, define $\delta(q, a)$ to equal $\diamondsuit$. Every string which was formerly undefined w.r.t. to δ^* is now mapped to $\diamondsuit$, a non-accepting state. This state is sometimes called the *sink* state or the *dead* state.

Definition 4. *A DFA is* complete *if δ is a total function. Otherwise it is* incomplete*.*

It is possible to write DFA which have many useless states. A state can be useless in two ways. First, there may be no string which forces the machine to transition into the state. Second, there may be a state from which no string

Definition 5. *A state q in a DFA A is* useful *if there is a string w such that $\delta^*(q_0, w) = q$ and a string v such that $\delta^*(q, v) \in F$. Otherwise q is* useless*. If every state in A is useful, then A is called* trim*.*

Not all complete DFAs are trim. If there is a sink state, it is useless in the above sense of the word.

Definition 6. *A DFA A is* minimal *if there is no other DFA A' such that $L(A) = L(A')$ and A' has fewer states than A.*

Technically, not all complete DFAs are minimal. If there is a sink state, it is not minimal.

Exercise 3. Consider the DFAs in the exercise 2. Are they complete? Trim? Minimal?

1.3 Some Closure Properties of Regular Languages

A set of objects X is *closed* under an operation $\circ$ if for all objects $x, y \in X$ it is the case that $x \circ y \in X$ too.

We can easily show that the union of any two regular stringsets R and S is also regular. Let $A_R = (Q_R, \Sigma, q_{0R}, F_R, \delta_R)$ be the DFA recognizing R and let $A_S = (Q_S, \Sigma, q_{0S}, F_S, \delta_S)$ be the DFA recognizing S. We can assume A_R and A_S are complete. We assume the same alphabet.

Construct $A = (Q, \Sigma, q_0, F, \delta)$ as follows.

- $Q = Q_R \times Q_S$.
- $q_0 = (q_{0R}, q_{0S})$.
- $F = \{(q_r, q_s) \mid q_r \in F_R \textbf{ or } q_s \in F_S\}$.
- $\delta((q_r, q_s), a) = (q_r', q_s')$ where $\delta_R(q_r, a) = q_r'$ and $\delta_S(q_s, a) = q_s'$.

Theorem 1. $L(A) = R \cup S$.

Similarly, the same kind of construction shows that the intersection of any two regular stringsets is regular. Construct $B = (Q, \Sigma, q_0, F, \delta)$ as follows.

- $Q = Q_R \times Q_S$.
- $q_0 = (q_{0R}, q_{0S})$.
- $F = \{(q_r, q_s) \mid q_r \in F_R$ **and** $q_s \in F_S\}$.
- $\delta((q_r, q_s), a) = (q'_r, q'_s)$ where $\delta_R(q_r, a) = q'_r$ and $\delta_S(q_s, a) = q'_s$.

Theorem 2. $L(B) = R \cap S$.

Here are some additional questions we are interested in for regular stringsets R and S.

1. Is the complement of R (denoted $\overline{R}$) a regular stringset?
2. Is $R \backslash S$ a regular stringset?
3. Can we decide whether $R \subseteq S$?
4. Is RS a regular stringset? (Note $RS = \{rs \mid r \in R$ and $s \in S\}$
5. Is R^* a regular stringset? (Note $R^0 = \{\lambda\}$, $R^n = R^{n-1}R$, $R^* = \bigcup_{n \in \mathbb{N}^0} R^n$)

The answers to all of these questions is Yes. With a little thought about complete DFA, the answers to first three follow very easily.

Theorem 3. *If R is a regular stringset then the complement of R is regular.*

Proof (Sketch). If R is a regular stringset then there is a complete DFA $A = (Q, \Sigma, q_0, F, \delta)$ which recognizes it. Let $B = (Q, \Sigma, q_0, F', \delta)$ where $F' = Q \backslash F$. We claim $L(B) = \overline{R}$. □

Corollary 1. *If R, S are regular stringsets then so is $R \backslash S$ since $R \backslash S = R \cap \overline{S}$.*

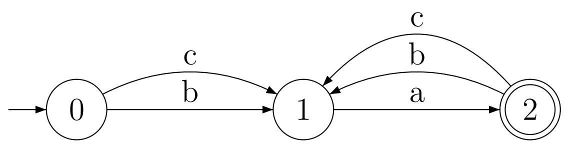
Corollary 2. *If R, S are regular stringsets then it is decidable whether $R \subseteq S$ since $R \subseteq S$ iff $R \backslash S = \varnothing$.*

Corollary 3. *If R, S are regular stringsets then it is decidable if $R = S$ since $R = S$ iff $R \subseteq S$ and $S \subseteq R$.*

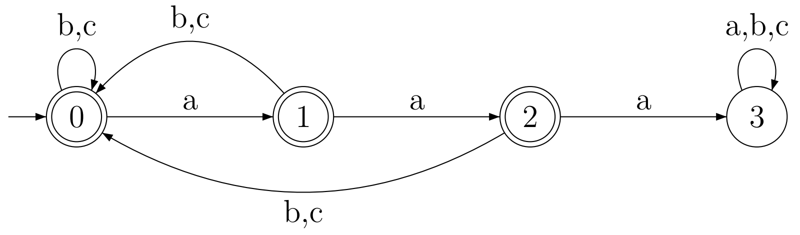
We postpone explaining how and why for the last two questions.

Some answers to Exercise 1

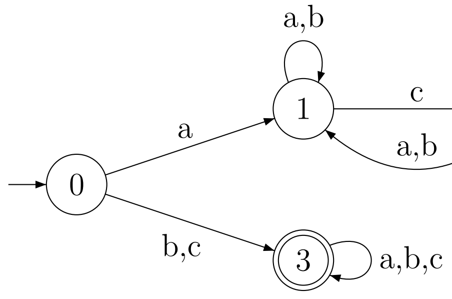
1.



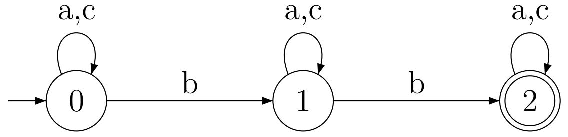
2.



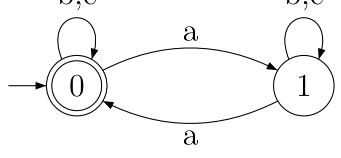
3.



4.



5.



Some answers to Exercise 2

1. Words must end in a consonant.
2. Words cannot contain both "b" and "c". So words permit only one type of consonant.
3. Words must have an odd number of vowels.

Chapter 2

String Transducers

2.1 Deterministic Finite-state String Transducers

2.1.1 Orientation

This section is about deterministic finite-state transducers for strings. The term *finite-state* means that the memory is bounded by a constant, no matter the size of the input to the machine. The term *deterministic* means there is single course of action the machine follows to compute the output from some input. The term *transducer* means this machine solves *transformation problems*: given an input object x, what object y is x transformed into? The term *string* means we are considering the transformation problem from strings to objects. So x is a string. As we will see, we can easily write transducers where y is a string, natural number, real number, or even a finite stringset! We will also see that DFSAs are a specific case of DFSTs.

However, we first define the output of the transformation to be a string. Then we will generalize it.

2.1.2 Definitions

Definition 7. *A* deterministic finite-state string-to-string transducer (DFST) *is a tuple* $T = (Q, \Sigma, M, q_0, v_0, \delta, \Omega, F)$ *where*

- Q *is a finite set of states;*
- Σ *is a finite set of symbols (*the input alphabet*);*
- μ *is a finite set of ouput symbols (*the output alphabet*);*
- $q_0 \in Q$ *is the* initial *state;*
- $v_0 \in \mu$ *is the* initial *value;*
- δ *is a function with domain* $Q \times \Sigma$ *and co-domain* Q*. It is called the* transition function.
- Ω *is a function with domain* $Q \times \Sigma$ *and co-domain* μ*. Let's call it the* output function.
- F *is a function with domain* Q *and co-domain* μ*. Let's call it the* final function.

As before we generalize the δ function to $\delta^* : Q \times \Sigma^* \to Q$. We also define a new function "process" $\pi : Q \times \mu^* \times \Sigma^* \to Q \times \mu^*$ as follows. π will process the string and produce its output.

$$\begin{aligned} \pi(q, v, \lambda) &= (q, v \cdot F(q)) \\ \pi^*(q, v, aw) &= \pi\big((\delta(q,a),\ v \cdot \Omega(q,a),\ w\big) \end{aligned} \tag{2.1}$$

Consider some DFST $T = (Q, \Sigma, q_0, F, \delta)$ and string $u \in \Sigma^*$. Then $T(u) = v$ where $(q, v) = \pi(q_0, v_0, u)$. We say T *transforms* u into v.

Definition 8. *The function defined by* T *is* $\big\{(u,v) \mid \exists q \in Q$ *such that* $\pi(q_0, v_0, u) = (q, v)\big\}$. *We write* $T(u) = v$ *iff* $(u, v) \in T$.

Definition 9. *A string-to-string function is called* sequential *if there is a DFST that recognizes it.*

2.1.3 Exercises

Exercise 4. Let $\Sigma = \mu = \{a, e, i, o, u, p, t, k, b, d, g, m, n, s, z, l, r\}$.

1. Write a transducer that prefixes *pa* to all words.
2. Write a transducer that suffixes *ing* to all words.
3. Write a transducer that deletes word initial vowels.
4. Write a transducer that voices obstruents which occur immediately after nasals.
5. Write a transducer that deletes word final vowels. So $T(abba) = abb$ and $T(pie) = pi$.
6. Write a transducer that voices obstruents intervocalically.

Note that the transition function and the final function can be partial functions. In this case, the transducer is *incomplete* in the sense it is not defined for all inputs. As before, we will strive to make our sequential transducers describe total functions.

2.2 Some Closure Properties of Sequential functions

Theorem 4 (Closure under composition)**.** *If* f, g *are sequential functions then so is* $f \circ g$.

Theorem 5 (Closure under intersection)**.** *If* f, g *are sequential functions then so is* $f \cap g$.

Theorem 6 (Minimal canonical form)**.** *For every sequential string-to-string function* f*, it is possible to compute a DFST* T *such that* T *is equivalent to* f *and no other DFST* T' *equivalent to* f *has fewer states than* T.

The closure theorems follow from proofs which use the product construction. The minimal cananoical form result is due to Choffrut (see his 2003 survey).

Sequential functions are not closed under union. This is because they are functions and so each input has a unique output.

2.3 Generalizing sequential functions with monoids

A *monoid* is a mathematical term which means any set which is closed under some associative binary operation with an identity. So if $(S, *, 1)$ is a monoid then for all $x, y \in S$:

1. is the case that $x * y$ is in S too (Closure under $*$)
2. $(x * y) * z = x * (y * z)$ (Associativity)
3. $1 * x = x * 1 = x$ (1 is the identity)

It is typical to refer to $*$ as "times", "multiplication" or as a product. It is also typical to refer to 1 as the "identity", "unit" or "one".

Σ^* is closed under the binary operation of concatenation. Also the empty string behaves like the identity with respect to concatenation. So $(\Sigma^*, \cdot, \lambda)$ is a monoid. As we processed the input string, we moved from state to state and updated the ouput value by concatenating strings along the output transitions. We can do the same thing and update the output value using some other product from another monoid.

Example 1. Here are some examples.

1. $(\{\texttt{True}, \texttt{False}\}, \wedge, \texttt{True})$. Boolean values and conjunction. This monoid shows the membership problem is a special case of the transformation problem.
2. $(\mathbb{N}, +, 0)$. Natural numbers and addition. Useful for counting!
3. $([0, 1], \times, 1)$. The real unit interval and multiplication. Useful for probabilities!
4. $(\mathbb{R}, \times, 1)$. All real numbers and multiplication.
5. $(\text{FIN}, \cdot, \{\lambda\})$ where FIN is the class of finite stringsets and $(\cdot)$ is concatenation of stringsets.
   - $\text{FIN} = \{S \mid \exists n \in \mathbb{N} \text{ with } |S| = n\}$
   - $S_1 \cdot S_2 = \{u \cdot v \mid u \in S_1, v \in S_2\}$.
6. There are many others!

This means we can generalize DFSTs to transducers which output elements from any monoid.

Definition 10. *A* generalized sequential transducer (GST) *is a tuple* $T = (Q, \Sigma, M, q_0, v_0, \delta, \Omega, F)$ *where*

- *Q is a finite set of states;*
- *Σ is a finite set of symbols (*the input alphabet*);*
- *$(\mu, *, 1)$ is a monoid*
- *$q_0 \in Q$ is the* initial *state;*
- *$v_0 \in \mu$ is the* initial *value;*
- *δ is a function with domain $Q \times \Sigma$ and co-domain Q. It is called the* transition function*.*

- Ω *is a function with domain* $Q \times \Sigma$ *and co-domain* μ*. Let's call it the* output function*.*
- F *is a function with domain* Q *and co-domain* μ*. Let's call it the* final function*.*

The process function π is the same except we replace concatenation of the outputs with the monoid operator $*$.

$$\begin{aligned} \pi(q, v, \lambda) &= (q, v * F(q)) \\ \pi^*(q, v, aw) &= \pi\big((\delta(q,a),\ v * \Omega(q,a),\ w\big) \end{aligned} \tag{2.2}$$

Then, the definition of the function computed by the transducer is identical to what was written formerly.

That's it!

2.3.1 Exercises

Exercise 5. Let $\Sigma = \mu = \{a, e, i, o, u, p, t, k, b, d, g, m, n, s, z, l, r\}$.

1. Write a transducer that counts how many NC (nasal-consonant) sequences occurs in the input word.
2. Write a transducer that optionally voices obstruents which occur immediately after nasals. So for in input like *anta* the output should be the set {*anda, anta*}. (Hint: use the FIN monoid).

2.4 Learning more

Unfortunately, the material on deterministic transducers has yet to make its way into standard textbooks. Standard textbooks in computer science discuss non-deterministic finite-state transducers if they discuss transducers at all. Within computational linguistics where transducers are widely used, non-determinstic ones are the norm. See, for instance Roche and Schabes (1997); Beesley and Kartunnen (2003) and Jurafsky and Martin (2008). A notable exception is work by Mehyar Mohri (1997; 2005). The most textbook-like discussion of sequential functions I am aware of comes from Lothaire (2005, chapter 1).

My interest in sequential transducers stems from three interrelated facts. First, they appear sufficient to decribe morpho-phonologial generalizations in natural language. So the extra power that comes with non-deteminsitic transducers appears unnecessary in this domain. Second, sequential transducers have canonical forms (Choffrut's theorem), but non-deterministic ones do not. Third, the class of sequential transducers can be learned from examples, unlike the class of non-deterministic transducers (de la Higuera, 2010, chapter 18).

Bibliography

Beesley, Kenneth, and Lauri Kartunnen. 2003. *Finite State Morphology*. CSLI Publications.

Choffrut, Christian. 2003. Minimizing subsequential transducers: a survey. *Theoretical Computer Science* 292:131 – 143.

de la Higuera, Colin. 2010. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press.

Jurafsky, Daniel, and James Martin. 2008. *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*. 2nd ed. Upper Saddle River, NJ: Prentice-Hall.

Lothaire, M., ed. 2005. *Applied Combinatorics on Words*. 2nd ed. Cmbridge University Press.

Mohri, Mehryar. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics* 23:269–311.

Mohri, Mehryar. 2005. Statistical natural language processing. In *Applied Combinatorics on Words*, edited by M. Lothaire. Cambridge University Press.

Roche, Emmanuel, and Yves Schabes. 1997. *Finite-State Language Processing*. MIT Press.