

Lecture Notes: Finite State Transducers

Instructor: *Daniele Micciancio*

UCSD CSE

This lecture notes are provided as a supplement to the textbook. In the exercises/problems section, the textbook defines Finite State Transducers (FST) as deterministic automata that each step read one input symbol $a \in \Sigma$ and output an output symbol $b \in \Gamma$. In Homework 1 you defined a more general form of FST that can output arbitrary strings at each step. (See homework solutions for a formal definition.) In these notes we consider further applications and extensions of FSTs.

1 FST Reductions

In the midterm we proved that regular languages are closed under the following operation: for any function $f_T: \Sigma^* \rightarrow \Gamma^*$ computed by an FST T , and any regular language $B \subseteq \Gamma^*$, the language $A = f_T^{-1}(B) = \{w \in \Sigma^* \mid f(w) \in B\}$ is regular. (As a reminder, this is proved by showing that any DFA for B can be combined with the FST T to obtain a DFA for A . Notice that, by definition, the requirement $A = f^{-1}(B)$ is equivalent to the condition “ $w \in A \iff f(w) \in B$ ”. (As usual, you can rewrite this double implication as two separate properties “ $w \in A \implies f(w) \in B$ ” and “ $w \notin A \implies f(w) \notin B$ ”.) A function $f: \Sigma^* \rightarrow \Gamma^*$ satisfying this property is called a *reduction* from A to B . For simplicity of exposition, below, we assume all languages are over some fixed alphabet Σ .

Definition 1. For any two languages $A, B \subseteq \Sigma^*$, a *reduction* from A to B is a function $f: \Sigma^* \rightarrow \Sigma^*$ such that for all $w \in \Sigma^*$,

- if $w \in A$ then $f(w) \in B$, and
- if $w \notin A$ then $f(w) \notin B$.

A reduction f is *FST-computable* if it is computed by a Finite State Transducer T . We say that A is *FST-reducible* to B (in symbols $A \leq_{FST} B$ if there is an FST-computable reduction f from A to B).

Reductions are one of the most important concepts in the study of the theory of computation, and we will encounter many other flavors of reductions later on. But for now, we will keep the discussion focused on regular languages. The notation $A \leq B$ (read “ A reduces to B ”) can be interpreted as a comparison between the “hardness” of the two problems. Think problems that can be solved by a DFA (i.e., regular languages) as being “easy”, and problems that cannot be solved by a DFA (i.e., nonregular languages) as being hard. The closure property proved in the midterm exam can be reformulated as follows.

Theorem 2. If $A \leq_{FST} B$ and B is regular, then A is also regular.

Informally, if A is not harder than B and B is “easy”, then you can conclude that A is also easy. By taking the contrapositive, we also get that if A is not harder than B and A is hard, then you can conclude that also B must be hard.

Corollary 3. If $A \leq_{FST} B$ and A is not regular, then B is also not regular.

Reductions are a powerful tool to study the complexity of computational problems, and can be used both to prove that certain problems are computationally easy, and others are computationally hard. But be careful: you need to use reductions in the correct direction. For example, if you show that $A \leq_{FST} B$ and also prove (or know) that A is regular, you cannot draw any conclusion about B : the problem B may be regular or nonregular. Similarly, if $A \leq_{FST} B$ and B is nonregular, you cannot conclude anything about the regularity of A .

2 Defining NFST

In applications it is sometime useful to consider nondeterministic transducers to model underspecified systems, user interaction, concurrency, etc. Here we provide a formal definition of nondeterministic finite state transducers (NFST), and prove that they are closed under composition. For an example of how NFST can be used in applications, see Homework 3.

Definition. A Nondeterministic Finite State Transducer (NFST) is a 5-tuple $M = (Q, \Sigma, \Gamma, \delta, s)$ consisting of

- A finite set of states Q
- A finite set of input symbols Σ
- A finite set of output symbols Γ
- A transition function $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q \times \Gamma^*)$
- A start state $s \in Q$

As all automata, at any given point during a computation, the internal state of an NFST is described by an element $q \in Q$ of the set of states. Initially the state is s , and it may change during the computation. During a computation, an NFST reads a string $w \in \Sigma^*$ over the input alphabet Σ , and outputs a string $u \in \Gamma^*$ over a possibly different alphabet Γ . The input symbols are read one at a time. When the machine is in state $q \in Q$ and reads the symbol $a \in \Sigma$, it selects (nondeterministically) an element $(p, w) \in \delta(q, a)$, updates its internal state to $p \in Q$ and prints $w \in \Gamma^*$. The output of the computation is obtained by concatenating the strings printed at every step. Since the machine is nondeterministic, there are several possible computations corresponding to the same input w , each producing a potentially different output string. So, the behavior of an NFST is described by a function $f_M : \Sigma^* \rightarrow \mathcal{P}(\Gamma^*)$ mapping the input string $w \in \Sigma^*$ to a set $f_M(w)$ of possible outputs. (This set can be empty if all computation branches abort.)

In order to formally define the output of an NFST, we first extend the transition function δ to a function $\delta_M^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q \times \Gamma^*)$ that can take strings as input, rather than single symbols.

Definition. Let $M = (Q, \Sigma, \Gamma, \delta, s)$ be an NFST. The extended transition function $\delta^*(q, w)$ is defined by induction on the length of w as follows:

- Base case ($|w| = 0$): for every $q \in Q$, let $\delta^*(q, \epsilon) = \{(q, \epsilon)\}$
- Inductive case ($|w| > 0$): for every $a \in \Sigma$ and $w' \in \Sigma^*$, let

$$\delta^*(q, aw') = \{(q'', u'u'') : \exists q' \in Q. (q', u') \in \delta(q, a) \wedge (q'', u'') \in \delta^*(q', w')\}$$

The set of possible outputs of M on input w is defined as

$$f_M(w) = \{u \in \Gamma^* : \exists q \in Q. (q, u) \in \delta^*(s, w)\}$$

i.e., the set of all strings that can be obtained starting from the initial state s and reading w .

3 Closure of NFST under composition

Each NFST describes a function $f : \Sigma^* \rightarrow \mathcal{P}(\Gamma^*)$, representing a (nondeterministic) system that on input a string $w \in \Sigma^*$ may produce one of many possible output strings $u \in f(w) \subseteq \Gamma^*$. Such functions can be composed together in a natural way: given $f : \Sigma^* \rightarrow \mathcal{P}(\Gamma^*)$ and $g : \Gamma^* \rightarrow \mathcal{P}(\Delta^*)$, the function composition of f and g is the function $g \circ f : \Sigma^* \rightarrow \mathcal{P}(\Delta^*)$ defined as $g \circ f(w) = g(f(w))$, i.e.,

$$g \circ f(w) = \{v \in \Delta^* : \exists u \in f(w). v \in g(u)\}.$$

In the homework you are asked to prove that (deterministic) FSTs are closed under composition, i.e., if f and g are computed by FSTs, then also the composite function $g \circ f$ is computed by an FST. Here we prove a similar result for NFST.

Theorem. *For any NFST $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, s_1)$ and $M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, s_2)$ with compatible alphabets $\Gamma_1 = \Sigma_2$, there is an NFST $M = M_2 \circ M_1$ such that $f_M = f_{M_2} \circ f_{M_1}$.*

Proof. The NFST M is defined as follows. Let $M = (Q, \Sigma_1, \Gamma_2, \delta, s)$ where $Q = Q_1 \times Q_2$, $s = (s_1, s_2) \in Q$ and $\delta: Q \times \Sigma_1 \rightarrow \mathcal{P}(Q \times \Gamma_2^*)$ is the function defined as

$$\delta((q_1, q_2), a) = \{((q'_1, q'_2), v) : \exists u \in \Gamma_1. (q'_1, u) \in \delta_1(q_1, a) \wedge (q'_2, v) \in \delta_2^*(q_2, u)\}.$$

It can be easily verified that $f_M = f_{M_2} \circ f_{M_1}$. □

A few words of explanation are due. The intuition behind the above construction is the following. The NFST $M_2 \circ M_1$ works by running M_1 on the input string $w \in \Sigma_1^*$ to obtain some intermediate result $u \in \Gamma_1^*$. The output alphabet Γ_1 is required to be the same as the input alphabet Σ_2 so that the output of M_1 can be fed as input to M_2 . As M_1 outputs w , the composed automaton $M_2 \circ M_1$ runs the second NFST on w to obtain the final output string v . Since finite automata (and NFST in particular) do not have enough memory to store the intermediate result of the computation w , the two component automata M_1, M_2 are run at the same time, and the output of M_1 is fed to M_2 while it is being produced. In order to run the two automata at the same time, we use the cartesian product $Q_1 \times Q_2$ as the set of states of the composite automaton. Each state $(q_1, q_2) \in Q$ records the current state of M_1 and the current state of M_2 . When a symbol $a \in \Sigma_1$ is read from the input, we first invoke the transition function of M_1 to obtain all possible actions $(q'_1, u) \in \delta_1(q_1, a)$ that M_1 can perform. Recall that (q'_1, u) means “move to state q'_1 and output u ”. Accordingly, the composite automaton runs M_2 on input u , starting from the current state q_2 , to obtain an updated state and output string $(q'_2, v) \in \delta_2^*(q_2, u)$. Notice that since the intermediate output $u \in \Gamma_1^*$ is not just a symbol, but a string, we need to use the extended transition function δ_2^* .