Before reading this, you should first study the previous notes on (deterministic) Finite State Transducers. In these notes we further extend the definition of FST with nondeterminism. Nondeterminism is important both in the theoretical study of automata, to better understand their power, and in applications where it can be used to model real life situations like underspecified systems, user interaction, concurrency, noisy communication channels, etc.

We start with the formal definition.

**Definition 1** A Nondeterministic Finite State Transducer (NFST) is defined by a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, s, F)$ where

- $Q$ is a finite set of states,
- $\Sigma$ is a finite set of input symbols,
- $\Gamma$ is a finite set of output symbols,
- $\delta : Q \times \Sigma \epsilon \rightarrow P(Q \times \Gamma \epsilon)$ is the transition function,
- $s \in Q$ is the start state, and
- $F \subseteq Q$ is a set of final states.

As a reminder, $\Sigma \epsilon = \Sigma \cup \{\epsilon\}$ is the extended input alphabet, which includes all alphabet symbols in $\Sigma$, and a special element $\epsilon$ denoting the empty string. Similarly, $\Gamma \epsilon$ is the extended output alphabet. We remark that this is just one possible formalization of NFSTs, and other definitions are possible. So, let us examine some of the choices made in this definition, and compare it to the definition of (deterministic) FSTs. $Q, \Sigma, \Gamma$ and $s$ are the same as before. The difference is in the type of the transition function $\delta$ and the introduction of a set of final states $F$. The definition extends that of deterministic FSTs in a way similar to how NFAs extend DFAs: the transition function $\delta$ outputs not just a single operation\(^1\) $(q, u) \in Q \times \Gamma^*$, but a (possibly empty) set of possible operations $\delta(q, a) \subseteq Q \times \Gamma^*$. Also, the transition function can be applied to the empty string $\delta(q, \epsilon)$, allowing the automaton to take a step without reading any input. At each step, the NFST selects (nondeterministically) one of the possible steps $(q, u) \in Q \times \Gamma^*$ from the output of the transition function, and executes it by transitioning to state $q$, and producing $u$ as partial output.

\(^1\)Notice that $\Gamma \epsilon$ can be regarded as the set of strings of length at most one, and therefore as a subset of $\Gamma^*$. We will elaborate on the use of $\Gamma \epsilon$ and $F$ shortly.
The reason we restrict the NFST to output at most one symbol per step is that allowing $\delta(q, a) \subseteq Q \times \Gamma^*$ would result in an automaton with a potentially infinite number of transitions out of each state. This technical issue could have been addressed by requiring $\delta(q, a)$ to be a finite subset of $Q \times \Gamma^*$. For simplicity, we chose to be even more stringent, and require the output of $\delta$ to be a subset of $Q \times \Gamma^*_\epsilon$, which is a finite set. The set of final states $F$ is introduced to make up for the limitation that the automaton can output at most one symbol at a time. Informally, a computation can terminate with an output string only if the NFST is in a state $q \in F$. If, upon reading a symbol $a$, the automaton wants to output a longer string $w = w_1 \ldots w_n$, it can do so by producing the symbols $w_i$ one at a time, while moving through a sequence of intermediate non-final states $q_i \in Q \setminus F$. After the last symbol $w_n$ is output, the automaton moves to a final state $q_n \in F$, and possibly terminates the computation.

Just like NFAs, an NFST can perform several different computations on a given input, and some of these computations may abort before the input has been completely processed. When a computation is aborted, its partial output accumulated during the computation is also discarded. The set of final states $F \subseteq Q$ gives more flexibility in selecting which computations abort: in order to complete a computation (and produce an output string) the NFST must reach a state in $F$ after reading the whole input. All this is defined more formally below.

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 $f_M(w) \subseteq \Gamma^*$ of possible output strings. (This set can be empty if all computation branches abort.) As usual, in order to formally define the output of an NFST, we first extend the transition function $\delta$ 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 2** Let $M = (Q, \Sigma, \Gamma, \delta, s, F)$ 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 $\delta^*(q, \epsilon)$ the set of all $(q', w) \in Q \times \Gamma^*$ such that for some $n \geq 0$, there exist $(q_i, w_i) \in \delta(q_{i-1}, \epsilon)$ (for $i = 1, \ldots, n$) such that $q_0 = q$, $q_n = q'$ and $w = w_1 \ldots w_n$.

- **Inductive case ($|w| > 0$):** for every $a \in \Sigma$ and $w' \in \Sigma^*$, let

$$\delta^*(q, aw') = \{(q'', u'u''u'''w'') \mid (q', w') \in \delta(q, a), (q'', u'') \in \delta(q', \epsilon), (q'''', u''') \in \delta(q'', w')\}.$$ 

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

$$f_M(w) = \{u \in \Gamma^* \mid (q, u) \in \delta^*(s, w), q \in F\}$$

i.e., the set of all strings that can be obtained starting from the initial state $s$, reading the input $w$, and ending in a state in $F$.

It follows from the previous discussion that, by introducing some additional (non-final) states, any FST can be transformed into an equivalent NFST.
Theorem 1  For any FST $M$ there is an NFST $N$ such that $f_N(w) = \{f_M(w)\}$ for every input $w$.

Notice that the converse is not true, because in general NFST can output several strings on a given input $w$, while FST always output only one string. So, nondeterminism in finite state transducers allows to describe a wider range of problems.

1 Closure properties and Reductions

Just as for FSTs, it is easy to prove that the class of functions computed by NFST is closed under composition. More precisely, the function $f_M: \Sigma^* \rightarrow \mathcal{P}(\Gamma^*)$ can be equivalently regarded as a relation $R_M = \{(x, y); y \in f_M(x)\}$. With some abuse of notation, we will refer to this relation also as $f_M$. Using the standard definition of composition for binary relations, for any $f: \Sigma^* \rightarrow \mathcal{P}(\Gamma^*)$ and $g: \Gamma^* \rightarrow \mathcal{P}(\Delta^*)$, we have

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

The following theorem shows that if $f$ and $g$ are NFST computable, then also their composition is computed by an NFST.

Theorem 2  For any NFST $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Gamma, \Delta, \delta_2, s_2, F_2)$, there is an NFST $M = M_2 \circ M_1$ such that $f_M = f_{M_2} \circ f_{M_1}$.

Proof The construction is similar to the deterministic case. Given $M_1$ and $M_2$, we combine them into a new NFST $M = (Q \times Q_2, \Sigma, \Delta, \delta, (s_1, s_2), F_1 \times F_2)$ where the transition function $\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q \times \Delta)$ is defined by

$$\delta((q_1, q_2), a) = \{((q'_1, q'_2), v) | (q'_1, u) \in \delta_1(q_1, a), (q'_2, v) \in \delta_2(q_2, u)\}.$$

□

It is also easy to show that any NFST $T$ can be combined with an NFA $M$, to obtain an NFA for the language $f_T^{-1}(L(M))$. This is similar to what we did in our study of FST reductions.

Theorem 3  For any NFST-computable function $f_T: \Sigma^* \rightarrow \Gamma^*$ and any regular language $B \subseteq \Gamma^*$, the language $A = f_T^{-1}(B) = \{w \in \Sigma^* | f(w) \in B\}$ is also regular.

Proof Let $M = (Q, \Sigma, \Gamma, \delta, s, F)$ be an NFA such that $L(M) = B$, and let $T = (Q_T, \Sigma, \Gamma, \delta_T, s_T)$ be an NFST. We combine $M$ and $T$ into an NFA $M' = (Q \times Q_T, \Sigma, \delta', (s, s_T), F \times F_T)$ where

$$\delta'((q, q_T), a) = \{(q', q_T') | (q_T, b) \in \delta_T(q_T, a), q' \in \delta(q, b)\}.$$

The language $f_T^{-1}(B)$ is regular because it is the language of the NFA $M'$.

□

The set of functions computable by an NFST is also closed under inversion.
**Theorem 4** For any NFST $M$, there is a corresponding NFST $M'$ such that

$$f_{M'}(y) = f^{-1}_M(y) = \{ x : y \in f_M(x) \}.$$  

**Proof** Let $M = (Q, \Sigma, \Gamma, \delta, s, F)$. The inverse NFST is defined as $M' = (Q \cup \{ s' \}, \Gamma, \delta', s', \{ s \})$ where $s'$ is a new start state, and the transition function is defined by the rules $\delta'(s', \epsilon) = F \times \{ \epsilon \}$ and $\delta'(q, a) = \{ (q', a') | (q, a) \in \delta(q', a') \}$.  

Intuitively, the above construction reverses the direction of all transitions, and swaps the start and final states. The new start state is introduced because our definition of NFST does not allow for multiple start states. Combining all the closure properties proved so far, it is easy to see that the image of a regular language under the function computer by an FST (or NFST) is also a regular languages.

**Corollary 1** For any FST or NFST $M$, if $A$ is a regular language, then also $f_M(A)$ is regular.

**Proof** If starting from an FST, first transform it into an equivalent NFST $M$. Then, by the closure property under inversion, there is some other NFST $M'$ such that $f_{M'} = f^{-1}_M$. This is equivalent to $f_M = f^{-1}_{M'}$. So, for any language $A$, we have $f_M(A) = f^{-1}_{M'}(A)$, i.e., there is an NFST reduction from $f_M(A)$ to $A$. It follows that if $A$ is regular, then also $f_M(A)$ is regular.