Nondeterministic Finite Automata

Nondeterministic Finite state automata differ from DFAs in that they allow for an input bit to specify multiple possible next moves. Namely, in a NFA we may move from state p to any of states $q_1, q_2, \ldots q_k$ by seeing the same input. Hence we will not have the same type of transition function as a DFA. Instead, our transition function will take as input a state and an element of the alphabet but return some subset of states. In an NFA, we also allow for multiple start states.

A *Nondeterministic Finite State Automata* is a model with 5 components:

$NFA = (Q, \Sigma, \Delta, S, F)$

$Q =$ a finite set, the *states*.

$\Sigma =$ a finite set, the *alphabet*

$\Delta =$ a function $Q \times \Sigma \to \mathcal{P}(Q)$, the *transition function*

$S =$ a subset of Q, the *start states*

$F =$ a subset of Q, the *final states*

Example Suppose we have an NFA with 3 states {home, cafe, school}. Let the input alphabet be $\{l, s\}$ and the transitions be as in the following diagram:

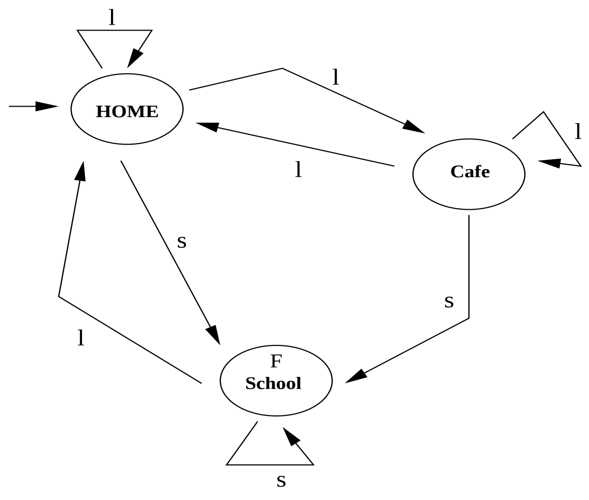

Figure 1: An NFA with 3 states.

We want to extend Δ over strings just as we extended δ for DFAs. In order to do this, we will also have to extend the first argument from Q to $\mathcal{P}(Q)$. This is because as we consider moving through the NFA as prescribed by some string, at any point we may be in one of many states. Formally we have:

$\hat{\Delta} : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ defined recursively by:

$\hat{\Delta}(P, \epsilon) = P$

$\hat{\Delta}(P, xa) = \cup_{q \in \hat{\Delta}(P,x)} \Delta(q, a)$

We say that an NFA *accepts* a string x if there is some way to move through it according to input x and end at a final state. Note that this just requires that there exists some way to reach a final state, there may also exist many ways not to reach the final state. We can write this as, the string x is accepted if $\hat{\Delta}(S, x) \cap F \neq \emptyset$. Just as for a DFA, the *language* is the set $\{x \in \Sigma^* | NFA \text{ accepts } x\}$.

Example What strings are accepted by the NFA in Figure 1? This model accepts strings which end in an s. What simpler model could we use to represent the same set of strings? We could remove the state cafe and all adjacent transitions.

Although it first it may seem that NFAs are more general, they actually have the exact same expressive power as DFAs. Namely, the sets of strings which are languages of NFAs are exactly the regular sets.

Clearly any DFA could be considered an NFA. In the example above, if we did remove the state cafe, the NFA would become a DFA. We will give a general construction to form a DFA from any NFA. This is known as the *subset construction*. Suppose we have an NFA $= (Q, \Sigma, \Delta, S, F)$. Form the DFA $= (\mathcal{P}(Q), \Sigma, \delta, s, F_D)$. The most important thing to notice is that the set of states of the DFA is the power set of states of the NFA. Now we may define $\delta(P, a) = \hat{\Delta}(P, a)$, $s = S$, and $F_D = \{P \subseteq Q | P \cap F \neq \emptyset\}$. Under this construction, the languages of the two models are the same. (why??)

Example Suppose we have the NFA of Figure 2. The subset construction would give the DFA of Figure 3.

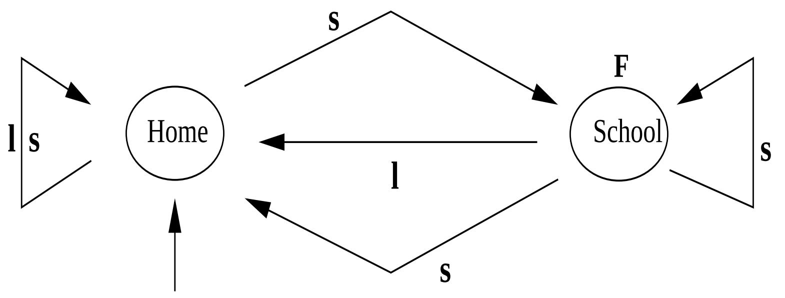

Figure 2: An NFA with 2 states.

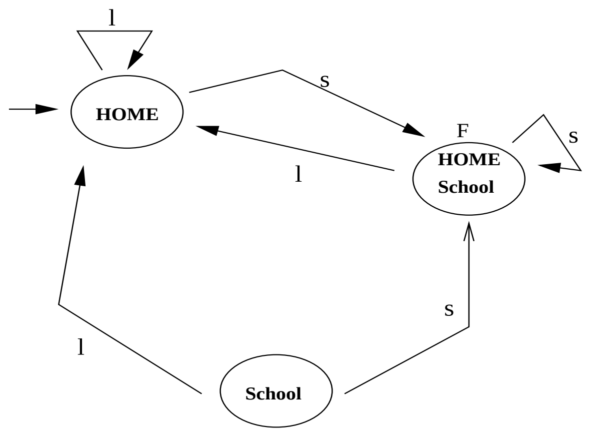

Figure 3: The DFA of the subset construction.

An ϵ-*transition* is a transition in either a DFA or an NFA which can be taken without seeing any input. It is something of a "free" transition. Again this may be a natural idea in many contexts but does not add any extra expression capability.