A deterministic finite automaton is a five-tuple

\[ M = (S, \Sigma, T, s_0, F) \]

where
- \( S \) is a finite set of "states",
- \( \Sigma \) is an alphabet — the "input alphabet",
- \( T : S \times \Sigma \rightarrow S \) is the "transition function",
- \( s_0 \in S \) is the "initial state",
- \( F \subset S \) is the set of "final" or "accepting" states.

We define the multi-step transition function

\[ T^* : S \times \Sigma^* \rightarrow S \]

as follows.

1. For any \( s \in S \), \( T^*(s, \Lambda) = s \).
2. For any \( s \in S \), \( x \in \Sigma^* \) and \( a \in \Sigma \),
   \[ T^*(s, xa) = T(T^*(s, x), a) \cdot \]

A string \( x \in \Sigma^* \) is accepted by \( M \) if

\[ T^*(s_0, x) \in F. \]

The language recognized by \( M \), denoted \( L(M) \), is the set of strings accepted by \( M \). That is,

\[ L(M) = \{ x \in \Sigma^* \mid T^*(s_0, x) \in F \}. \]
Also recall  The set of regular languages over an alphabet $\Sigma$ is the least set of languages over $\Sigma$ s.t.
1. $\emptyset$ is a regular language,
2. $\{\Lambda\}$ is a regular language,
3. For all $a \in \Sigma$, $\{a\}$ is a regular language,
4. If $A$ is a regular language, so is $A^*$,
5. If $A$ and $B$ are regular languages, so are $A \cup B$ and $AB$.

And the set of regular expressions is similarly defined, so that it is immediately clear that every regular expression stands for a (specific) regular language, and every regular language is represented by some regular expression.

Is every regular language represented by more than one regular expression?

For a given regular language $L$, how many regular expressions stand for $L$?

How many regular languages are there (for a given alphabet $\Sigma$)? (Related question: How many regular expressions are there?)
Complement construction for DFA’s

**Observation**  By the inductive definition of regular languages, we see that the set of regular languages is closed under (finite) union, product and Kleene closure.

**Theorem**  The regular languages are closed under set complement.

**Proof.**  Consider any regular language \( L \) over \( \Sigma \). By Kleene’s Theorem, \( L \) is accepted by some DFA

\[
M = (S, \Sigma, T, s_0, F).
\]

Let

\[
M' = (S, \Sigma, T, s_0, S - F).
\]

DFA’s \( M \) and \( M' \) differ only on their sets of accepting states, which are complements. In particular, both DFA’s have the same multi-step transition function \( T^* : S \times \Sigma^* \rightarrow S \). Hence, for any \( x \in \Sigma^* \),

\[
M \text{ accepts } x \text{ iff } T^*(s_0, x) \in F
\]

while

\[
M' \text{ accepts } x \text{ iff } T^*(s_0, x) \notin F.
\]

That is, \( M \) accepts \( x \) iff \( M' \) doesn’t. So we see that \( M' \) accepts the language \( L' \) (that is, \( \Sigma^* - L \)). By Kleene’s Theorem we can conclude that the regular languages are closed under set complement.
Closure under intersection and set difference

**Theorem**  The regular languages are closed under set intersection and set difference.

*Proof.* We know (by the inductive defn) that the regular languages are closed under union, and we have shown (by DFA construction) that the regular languages are also closed under complement. Since

\[ A \cap B = (A' \cup B')' \]

and

\[ A - B = A \cap B' , \]

we can conclude that the set of regular languages is also closed under set intersection and set difference.
Question  Given DFA’s $M_1$ and $M_2$, can we construct DFA’s to accept

- $L(M_1) \cup L(M_2)$?
- $L(M_1)L(M_2)$?
- $L(M_1)^*$?
- $L(M_1) \cap L(M_2)$?
- $L(M_1) - L(M_2)$?

The short answer is “yes” for all of these. (And the ability to do the first three is crucial to the argument showing that every regular language is accepted by some DFA!)

For product and Kleene closure we will wait until we have a “generalized” version of DFA’s that makes the constructions easier.

But the construction for union is already easy, and by slightly altering this construction, we obtain DFA’s for $L_1 \cap L_2$ and $L_1 - L_2$ also...
Example  Before looking at the general construction, let’s try an example. Given DFA’s for \( \{0\}^*\{1\}^* \) and \( \{1\}^*\{0\}^* \), construct a DFA for the intersection of the two languages...
DFA construction for union, intersection, and set difference

Theorem  Given DFA's

\[ M_1 = (S_1, \Sigma, T_1, s_1, F_1) \quad \text{and} \quad M_2 = (S_2, \Sigma, T_2, s_2, F_2), \]

let

\[ M = (S, \Sigma, T, s_0, F) \]

where

- \( S = S_1 \times S_2 \),
- \( s_0 = (s_1, s_2) \), and
- for all \((s, s') \in S\) and \(a \in \Sigma\),
  \[ T((s, s'), a) = (T_1(s, a), T_2(s', a)). \]

Then the following hold.

1. If \( F = \{ (s, s') \in S \mid s \in F_1 \text{ or } s' \in F_2 \} \), then
  \[ L(M) = L(M_1) \cup L(M_2). \]

2. If \( F = \{ (s, s') \in S \mid s \in F_1 \text{ and } s' \in F_2 \} \), then
  \[ L(M) = L(M_1) \cap L(M_2). \]

3. If \( F = \{ (s, s') \in S \mid s \in F_1 \text{ and } s' \notin F_2 \} \), then
  \[ L(M) = L(M_1) - L(M_2). \]
Note: This construction may produce states in $M$ that are not reachable from the start state, but it is easy to show that (in any DFA) unreachable states can be dropped (without affecting the strings that are accepted).

BTW How can we use our notation to express that a state in DFA $M$ is reachable (from the start state)?
DFA for the intersection of \( \{0, 11\}^* \) and \( \{00, 1\}^* \)
\[ T^* : S \times \Sigma^* \rightarrow S \]
\[
T^*(s, \Lambda) = s, \text{ for all } s \in S \\
T^*(s, xa) = T(T^*(s, x), a), \text{ for all } s \in S, x \in \Sigma^*, \text{ and } a \in \Sigma
\]

**Lemma** For any DFA's \( M_1 = (S_1, \Sigma, T_1, s_1, F_1) \), \( M_2 = (S_2, \Sigma, T_2, s_2, F_2) \), let
\[
M = (S, \Sigma, T, s_0, F)
\]
where \( S = S_1 \times S_2 \), \( s_0 = (s_1, s_2) \), and for all \( (s, s') \in S \) and \( a \in \Sigma \),
\[
T((s, s'), a) = (T_1(s, a), T_2(s', a)).
\]

For all \( (s, s') \in S \) and \( x \in \Sigma^* \),
\[
T^*((s, s'), x) = (T_1^*(s, x), T_2^*(s', x)).
\]

**Proof.** By structural induction on \( x \).

**Basis:**
\[
T^*((s, s'), \Lambda) = (s, s') \quad \text{(defn } T^*)
\]
\[
= (T_1^*(s, \Lambda), T_2^*(s', \Lambda)) \quad \text{(defn } T_1^*, \text{ defn } T_2^*)
\]

**Induction:** \( x \in \Sigma^*, a \in \Sigma \).

**IH:** \( T^*((s, s'), x) = (T_1^*(s, x), T_2^*(s', x)) \).

**NTS:** \( T^*((s, s'), xa) = (T_1^*(s, xa), T_2^*(s', xa)) \).

\[
T^*((s, s'), xa) = T(T^*((s, s'), x), a) \quad \text{(defn } T^*)
\]
\[
= T((T_1^*(s, x), T_2^*(s', x)), a) \quad \text{(IH)}
\]
\[
= (T_1(T_1^*(s, x), a), T_2(T_2^*(s', x), a)) \quad \text{(defn } T)
\]
\[
= (T_1^*(s, xa), T_2^*(s', xa)) \quad \text{(defn } T_1^*, \text{ defn } T_2^*)
\]
Proof for DFA union/intersection/difference construction

Proof. For part 1, we must show that, for every $x \in \Sigma^*$,

$M$ accepts $x$ iff $M_1$ or $M_2$ does.

By the definition of $M$ in part 1, $M$ accepts $x$ iff

$$T^*((s_1, s_2), x) \in \{ (s, s') \in S \mid s \in F_1 \text{ or } s' \in F_2 \}.$$ 

By the Lemma

$$T^*((s_1, s_2), x) = (T_1^*(s_1, x), T_2^*(s_2, x)).$$

So

$M$ accepts $x$ iff $T_1^*(s_1, x) \in F_1$ or $T_2^*(s_2, x) \in F_2$.

Which is to say that $M$ accepts $x$ iff $M_1$ or $M_2$ does.

Proofs for parts 2 and 3 are similar.
Question  Given DFA’s $M_1$ and $M_2$, can we construct DFA’s to accept

- $L(M_1)'$?
- $L(M_1) \cup L(M_2)$?
- $L(M_1) \cap L(M_2)$?
- $L(M_1) - L(M_2)$?
- $L(M_1)L(M_2)$?
- $L(M_1)^*$?

We already have nice DFA constructions for complement, union, intersection, and difference.

For product and Kleene closure we want a “generalized” version of DFA’s that makes the constructions easier.

First, we’ll add “nondeterminism”...
**Example** Recall the family of languages

\[ L_n = (0 + 1)^*1(0 + 1)^n. \]

We used the Distinguishability Theorem to show that any DFA for \( L_n \) must have at least \( 2^{n+1} \) states.

What about the following “nondeterministic” machine for

\[ (0 + 1)^*1(0 + 1)^2? \]

Is it a DFA?

But it can be understood to accept \( (0 + 1)^*1(0 + 1)^2 \), and does this using only 4 states. (And in general, it appears that for each language \( L_n \) there should be such a machine with only \( n + 2 \) states!)
NFA’s

A *nondeterministic finite automaton* (NFA) is a 5-tuple

\[(S, \Sigma, T, s_0, F)\]

where

- \(S\) is a finite set of “states”,
- \(\Sigma\) is an “input” alphabet,
- \(T : S \times \Sigma \rightarrow \text{power}(S)\) is the “transition function”,
- \(s_0 \in S\) is the “initial state”, and
- \(F \subset S\) is the set of “final” or “accepting” states.

So the definition of an NFA is the same as for a DFA, except for the transition function \(T\), which now takes a state and a symbol to a set of states (instead of a single state).

Did our diagram for \((0 + 1)^*1(0 + 1)^2\) fit this definition?
**Definition of $T^*$ for NFA’s**

As with DFA’s, we need to define a multi-step transition function. But in this case it is a little harder, since reading a string in an NFA may take us to a number of different states (i.e. a set of states).

Recall that the definition of $T^*$ for DFA’s is recursive. The base case says

$$T^*(s, \Lambda) = s$$

and the recursive equation says what to do for (nonempty) strings $xa$, in terms of what to do on $x$ and what to do on $a$.

We do the same sort of thing for NFA’s.

**Definition** Given an NFA $M = (S, \Sigma, T, s_0, F)$, we define the multi-step transition function

$$T^* : S \times \Sigma^* \to \text{power}(S)$$

as follows.

1. For all $s \in S$, $T^*(s, \Lambda) = \{s\}$.
2. For all $s \in S$, $x \in \Sigma^*$ and $a \in \Sigma$, $T^*(s, xa) = \bigcup_{s' \in T^*(s, x)} T(s', a)$. 
Some properties of $T^*$ for NFA’s that one might want to verify…

\[ T^*(s, a) = T(s, a) \]

\[ T^*(s, xy) = \bigcup_{s' \in T^*(s, x)} T^*(s', y) \]

If $T^*(s, x) = \emptyset$, then $T^*(s, xy) = \emptyset$. 
Acceptance for NFA’s

Roughly:

An NFA $M$ accepts $x$ if there is a sequence of moves $M$ can make on input $x$ that ends in an accepting state.

Precisely:

**Definition** Given an NFA $M = (S, \Sigma, T, s_0, F)$ and a string $x \in \Sigma^*$, $M$ accepts $x$ if

$$T^*(s_0, x) \cap F \neq \emptyset.$$

As before, the language *recognized* by $M$, denoted $L(M)$, is the set of strings accepted by $M$. That is,

$$L(M) = \{ x \in \Sigma^* | T^*(s_0, x) \cap F \neq \emptyset \}.$$
Reducing NFA’s to DFA’s

**Example**  Any NFA can be reduced to a DFA that recognizes the same language. Before we look at the general ("subset") construction, let’s try an example. Consider again the NFA for

\[(0 + 1)^* 1(0 + 1)^2.\]
**Subset Construction Theorem**  Given an NFA

\[ M = (S, \Sigma, T, s_0, F) , \]

let DFA

\[ M_d = (\text{power}(S), \Sigma, T_d, \{s_0\}, F_d) \]

be s.t.

- for every \( D \in \text{power}(S) \) and \( a \in \Sigma \),

\[ T_d(D, a) = \bigcup_{s \in D} T(s, a) \]

- \( F_d = \{ D \in \text{power}(S) \mid D \cap T \neq \emptyset \} \).

Then \( L(M_d) = L(M) \).

**Proof.** We will first show that for all \( x \in \Sigma^* \),

\[ T_d^* (\{s_0\}, x) = T^* (s_0, x) \]

by structural induction on \( x \). Notice that \( T_d^* \) and \( T^* \) are defined differently: the first belongs to a DFA, the second to an NFA...
Given NFA $M = (S, \Sigma, T, s_0, F)$, we construct DFA $M_d = (\text{power}(S), \Sigma, T_d, \{s_0\}, F_d)$, where

$F_d = \{ D \in \text{power}(S) \mid D \cap F \neq \emptyset \}$, and for every $D \in \text{power}(S)$ and $a \in \Sigma$,

$$T_d(D, a) = \bigcup_{s \in D} T(s, a).$$

According to defn of multi-step transition function for DFA’s:

1. For any $D \in \text{power}(S)$, $T^*_d(D, \Lambda) = D$.
2. For any $D \in \text{power}(S)$, $x \in \Sigma^*$ and $a \in \Sigma$,

$$T^*_d(D, xa) = T_d(T^*_d(D, x), a).$$

NFA’s defn of $T^*$:

1. For all $s \in S$, $T^*(s, \Lambda) = \{s\}$.
2. For all $s \in S$, $x \in \Sigma^*$ and $a \in \Sigma$,

$$T^*(s, xa) = \bigcup_{s' \in T^*(s, x)} T(s', a).$$

We will show by structural induction that for all $x \in \Sigma^*$,

$$T^*_d(\{s_0\}, x) = T^*(s_0, x).$$

Basis: We need to show that $T^*_d(\{s_0\}, \Lambda) = T^*(s_0, \Lambda)$.

$$T^*_d(\{s_0\}, \Lambda) = \{s_0\} \quad \text{(defn } T^*_d \text{ for A)}$$

$$= T^*(s_0, \Lambda) \quad \text{(defn } T^* \text{ for NFA)}$$
Given NFA $M = (S, \Sigma, T, s_0, F)$, we construct DFA $M_d = (\text{power}(S), \Sigma, T_d, \{s_0\}, F_d)$, where $F_d = \{ D \in \text{power}(S) \mid D \cap F \neq \emptyset \}$, and for every $D \in \text{power}(S)$ and $a \in \Sigma$,

$$T_d(D, a) = \bigcup_{s \in D} T(s, a).$$

According to defn of multi-step transition function for DFA's:

1. For any $D \in \text{power}(S)$, $T_d^*(D, \Lambda) = D$.
2. For any $D \in \text{power}(S)$, $x \in \Sigma^*$ and $a \in \Sigma$, $T_d^*(D, xa) = T_d(T_d^*(D, x), a)$.

NFA’s defn of $T^*$:

1. For all $s \in S$, $T^*(s, \Lambda) = \{s\}$.
2. For all $s \in S$, $x \in \Sigma^*$ and $a \in \Sigma$, $T^*(s, xa) = \bigcup_{s' \in T^*(s, x)} T(s', a)$.

---

**Induction:** $x \in \Sigma^*$, $a \in \Sigma$.

IH: $T_d^*(\{s_0\}, x) = T^*(s_0, x)$.

NTS: $T_d^*(\{s_0\}, xa) = T^*(s_0, xa)$.

$$T_d^*(\{s_0\}, xa) = T_d(T_d^*(\{s_0\}, x), a) \quad \text{(defn $T_d^*$ for DFA)}$$

$$= T_d(T^*(s_0, x), a) \quad \text{(IH)}$$

$$= \bigcup_{s \in T^*(s_0, x)} T(s, a) \quad \text{(def $T_d$)}$$

$$= T^*(s_0, xa) \quad \text{(defn $T^*$ for NFA)}$$

So we conclude that for all $x \in \Sigma^*$, $T_d^*(\{s_0\}, x) = T^*(s_0, x)$. 
NFA $M = (S, \Sigma, T, s_0, F)$.  
FA $M_d = (\text{power}(S), \Sigma, T_d, \{s_0\}, F_d)$, where $F_d = \{ D \in \text{power}(S) \mid D \cap F \neq \emptyset \}$, and for every $D \in \text{power}(S)$ and $a \in \Sigma$, 

$$T_d(D, a) = \bigcup_{s \in D} T(s, a).$$

So we’ve established that for all $x \in \Sigma^*$, 

$$T_d^* (\{s_0\}, x) = T^* (s_0, x).$$

It remains only to use this fact to show that, for all $x \in \Sigma^*$, $M_d$ accepts $x$ iff $M$ does.

$M_d$ accepts $x$

iff $T_d^* (\{s_0\}, x) \in F_d$ (defn acceptance for DFA $M_d$)

iff $T_d^* (\{s_0\}, x) \cap F \neq \emptyset$ (defn of $F_d$ in terms of $F$)

iff $T^* (s_0, x) \cap F \neq \emptyset$ ($T_d^* (\{s_0\}, x) = T^* (s_0, x)$)

iff $M$ accepts $x$ (defn acceptance for NFA $M$)
Reducing an NFA for $0^+ + 1(0 + 1)^*0$ to a DFA
NFA’s still aren’t convenient enough. . .

It is generally easier to “capture” a regular expression in an NFA. After all, a key feature of regular expressions is nondeterminism.

But to prove that every regular language is accepted by some NFA (and so by some DFA), we need a general construction for union, product and Kleene closure. This is awkward to define directly for NFA’s.

The usual next step is to extend NFA’s by also allowing so-called “Λ transitions”. We call the resulting machines “NFA-Λ’s”.

Λ transitions make it possible for an NFA-Λ to change state without reading a symbol, which turns out to be extremely convenient. Intuitively, such state transitions can capture the idea of making a choice when generating a string from a regular expression.

NFA-Λ’s can be reduced to NFA’s (although we won’t learn how to do this). And since NFA’s can be reduced in turn to DFA’s, we know that any language recognized by an NFA-Λ is also recognized by some DFA.

And for NFA-Λ’s, there are easy constructions for union, product and Kleene closure! So it is easy to prove that every regular language is recognized by some NFA-Λ.

We’ll look at a couple of examples of the use of NFA-Λ’s before defining them. . .
Product made easy (by allowing \( \Lambda \) transitions)

Consider \((0^* + 1^*)(0^+ + 1^+)\):
Kleene closure made easy (by allowing \( \Lambda \) transitions)

Consider \((00)^* + (11)^+)^*:\)
NFA’s with \( \Lambda \) transitions

**Definition** A *nondeterministic finite automaton with \( \Lambda \) transitions* (NFA-\( \Lambda \)) is a 5-tuple \((S, \Sigma, T, s_0, F)\) where

- \( S \) is a finite set of “states”,
- \( \Sigma \) is the “input” alphabet,
- \( T : S \times (\Sigma \cup \{\Lambda\}) \to \text{power}(S) \) is the “transition function”,
- \( s_0 \in S \) is the “initial state”,
- \( F \subset S \) is the set of “final” or “accepting” states.

Once again, we would want to define a multi-step transition function. But consider the kind of difficulty we face:

For NFA’s we had the base case \( T^*(s, \Lambda) = \{s\} \).

For NFA-\( \Lambda \)’s it is clear that we still have \( s \in T^*(s, \Lambda) \), but it is also clear that there may be other states “reachable” from \( s \) on the empty string — states reachable from \( s \) by one or more \( \Lambda \)-transitions.

We won’t grapple with this difficulty in this course. (The set of states reachable from a given state by \( \Lambda \)-transitions is an inductively defined set. . . )
Equivalence of DFA’s, NFA’s and NFA-Λ’s

**Theorem**  For any alphabet $\Sigma$ and any language $L \subseteq \Sigma^*$, the following are equivalent:

- There is a DFA that recognizes $L$.
- There is an NFA that recognizes $L$.
- There is an NFA-Λ that recognizes $L$.
Proof sketch for half of Kleene’s Theorem

**Lemma** Every regular language is recognized by some DFA.

*Proof sketch.* By the previous theorem, any language recognized by some NFA-Λ is recognized by some DFA. So it is enough to show that every regular language is recognized by some NFA-Λ.

Proof is by structural induction on the inductive definition of the set of regular languages over $\Sigma$.

*Case 1:* There is an NFA-Λ that recognizes $\emptyset$.

*Case 2:* There is an NFA-Λ that recognizes $\{\Lambda\}$.

*Case 3:* For every $a \in \Sigma$, there is an NFA-Λ that recognizes $\{a\}$.

*Case 4:* $L$ is a regular language over $\Sigma$.

  *IH:* There is an NFA-Λ that recognizes $L$.

  *NTS:* There is an NFA-Λ that recognizes $L^*$.

*Case 5:* $L_1, L_2$ are regular languages over $\Sigma$.

  *IH:* There are NFA-Λ’s that recognize $L_1$ and $L_2$.

  *NTS:* There are NFA-Λ’s that recognize $L_1 \cup L_2$ and $L_1L_2$. 