3.11: Deterministic Finite Automata

In this section, we study the third of our more restricted kinds of finite automata: deterministic finite automata.

Definition of DFAs

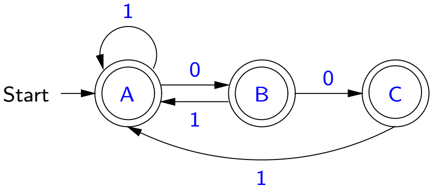
A *deterministic finite automaton* (DFA) M is a finite automaton such that:

- $T_M \subseteq \{\, q, x \to r \mid q, r \in \textbf{Sym} \text{ and } x \in \textbf{Str} \text{ and } |x| = 1 \,\}$; and
- for all $q \in Q_M$ and $a \in \textbf{alphabet}\, M$, there is a unique $r \in Q_M$ such that $q, a \to r \in T_M$.

We write **DFA** for the set of all deterministic finite automata. Thus $\textbf{DFA} \subsetneq \textbf{NFA} \subsetneq \textbf{EFA} \subsetneq \textbf{FA}$.

Example DFA

Let M be the finite automaton

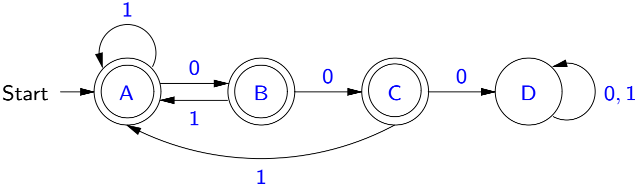


Then $L(M) = \{ w \in \{0,1\}^* \mid 000 \text{ is not a substring of } w \}$.

Is M a DFA? No. M is an NFA. But $0 \in$ **alphabet** M and there is no transition of the form $\mathsf{C}, 0 \to r$.

Example DFA

We can make M into a DFA by adding a dead state D:



We will never need more than one dead state in a DFA.

Properties of DFAs

The following proposition obviously holds.

Proposition 3.11.1
Suppose M is a DFA.

- *For all $N \in$ **FA**, if M **iso** N, then N is a DFA.*
- *For all bijections f from Q_M to some set of symbols, **renameStates**(M, f) is a DFA.*
- ***renameStatesCanonically** M is a DFA.*

The δ Function

Proposition 3.11.2
Suppose M is a DFA. For all $q \in Q_M$ and $w \in (\textbf{alphabet}\ M)^*$, $|\Delta_M(\{q\}, w)| = 1$.

Proof. An easy left string induction on w. □

Because of Proposition 3.11.2, we can define *the transition function δ_M for M*, $\delta_M \in Q_M \times (\textbf{alphabet}\ M)^* \to Q_M$, by:

$$\delta_M(q, w) = \text{the unique } r \in Q_M \text{ such that } r \in \Delta_M(\{q\}, w).$$

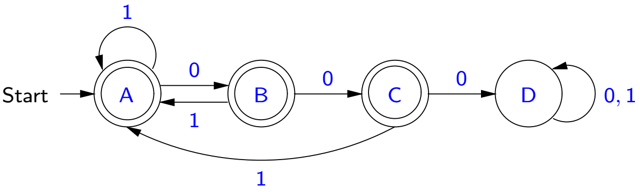
Thus, for all $q, r \in Q_M$ and $w \in (\textbf{alphabet}\ M)^*$,

$$\delta_M(q, w) = r \qquad \text{iff} \qquad r \in \Delta_M(\{q\}, w).$$

We sometimes abbreviate $\delta_M(q, w)$ to $\delta(q, w)$.

Example Uses of δ

For example, if M is the DFA



then

- $\delta(\mathsf{A}, \%) = \mathsf{A}$;
- $\delta(\mathsf{A}, 0100) = \mathsf{C}$;
- $\delta(\mathsf{B}, 000100) = \mathsf{D}$.

Properties of δ

Proposition 3.11.3
Suppose M is a DFA.

(1) *For all $q \in Q_M$, $\delta_M(q, \%) = q$.*

(2) *For all $q \in Q_M$ and $a \in$ **alphabet** M, $\delta_M(q, a) =$ the unique $r \in Q_M$ such that $q, a \to r \in T_M$.*

(3) *For all $q \in Q_M$ and $x, y \in (\textbf{alphabet}\, M)^*$, $\delta_M(q, xy) = \delta_M(\delta_M(q, x), y)$.*

Suppose M is a DFA. By part (2) of the proposition, we have that, for all $q, r \in Q_M$ and $a \in$ **alphabet** M,

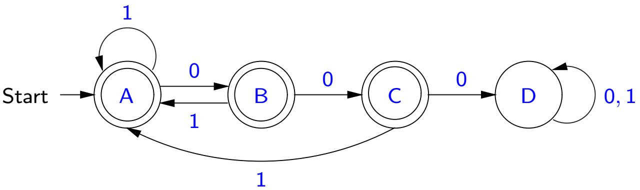
$$\delta_M(q, a) = r \qquad \text{iff} \qquad q, a \to r \in T_M.$$

Checking Acceptance in DFAs

Proposition 3.11.4

Suppose M is a DFA.
$L(M) = \{\, w \in (\textbf{alphabet}\, M)^* \mid \delta_M(s_M, w) \in A_M \,\}$.

The preceding propositions give us an efficient algorithm for checking whether a string is accepted by a DFA. For example, suppose M is the DFA



To check whether 0100 is accepted by M, we need to determine whether $\delta(\mathsf{A}, 0100) \in \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$.

Checking Acceptance in DFAs

We have that:

$$\begin{aligned}
\delta(\mathsf{A}, 0100) &= \delta(\delta(\mathsf{A}, 0), 100) \\
&= \delta(\mathsf{B}, 100) \\
&= \delta(\delta(\mathsf{B}, 1), 00) \\
&= \delta(\mathsf{A}, 00) \\
&= \delta(\delta(\mathsf{A}, 0), 0) \\
&= \delta(\mathsf{B}, 0) \\
&= \mathsf{C} \\
&\in \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}.
\end{aligned}$$

Thus 0100 is accepted by M.

Proving the Correctness of DFAs

Since every DFA is an FA, we could prove the correctness of DFAs using the techniques that we have already studied.

But it turns out that giving a separate proof that enough is accepted by a DFA is unnecessary—it will follow from the proof that everything accepted is wanted.

Proposition 3.11.5
Suppose M *is a DFA. Then, for all* $w \in (\textbf{alphabet}\, M)^*$ *and* $q \in Q_M$,

$$w \in \Lambda_{M,q} \quad \textit{iff} \quad \delta_M(s_M, w) = q.$$

We already know that, if M is an FA, then $L(M) = \bigcup\{\, \Lambda_q \mid q \in A_M \,\}$.

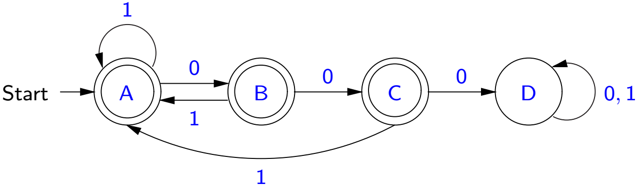
Proposition 3.11.6
Suppose M *is a DFA.*

(1) $(\textbf{alphabet}\, M)^* = \bigcup\{\, \Lambda_q \mid q \in Q_M \,\}$.

(2) For all $q, r \in Q_M$, *if* $q \neq r$, *then* $\Lambda_q \cap \Lambda_r = \emptyset$.

Example DFA Correctness Proof

Suppose M is the DFA



and let $X = \{\, w \in \{0,1\}^* \mid 000 \text{ is not a substring of } w \,\}$.

We will show that $L(M) = X$.

Note that, for all $w \in \{0,1\}^*$:

- $w \in X$ iff 000 is not a substring of w; and
- $w \notin X$ iff 000 is a substring of w.

Example Correctness Proof

First, we use induction on Λ, to prove that:

(A) for all $w \in \Lambda_{\mathsf{A}}$, $w \in X$ and 0 is not a suffix of w;

(B) for all $w \in \Lambda_{\mathsf{B}}$, $w \in X$ and 0, but not 00, is a suffix of w;

(C) for all $w \in \Lambda_{\mathsf{C}}$, $w \in X$ and 00 is a suffix of w;

(D) for all $w \in \Lambda_{\mathsf{D}}$, $w \notin X$.

There are nine steps (1 + the number of transitions) to show.

- **(empty string)** We must show that $\% \in X$ and 0 is not a suffix of $\%$. This follows since $\%$ has no 0's.
- **($\mathsf{A}, 0 \rightarrow \mathsf{B}$)** Suppose $w \in \Lambda_{\mathsf{A}}$, and assume the inductive hypothesis: $w \in X$ and 0 is not a suffix of w. We must show that $w0 \in X$ and 0, but not 00, is a suffix of $w0$. Because $w \in X$ and 0 is not a suffix of w, we have that $w0 \in X$. Clearly, 0 is a suffix of $w0$. And, since 0 is not a suffix of w, we have that 00 is not a suffix of $w0$.

Example Correctness Proof

- **(**$\mathsf{A}, 1 \to \mathsf{A}$**)** Suppose $w \in \Lambda_{\mathsf{A}}$, and assume the inductive hypothesis: $w \in X$ and 0 is not a suffix of w. We must show that $w1 \in X$ and 0 is not a suffix of $w1$. Since $w \in X$, we have that $w1 \in X$. And, 0 is not a suffix of $w1$.
- **(**$\mathsf{B}, 0 \to \mathsf{C}$**)** Suppose $w \in \Lambda_{\mathsf{B}}$, and assume the inductive hypothesis: $w \in X$ and 0, but not 00, is a suffix of w. We must show that $w0 \in X$ and 00 is a suffix of $w0$. Because $w \in X$ and 00 is not suffix of w, we have that $w0 \in X$. And since 0 is a suffix of w, it follows that 00 is a suffix of $w0$.
- **(**$\mathsf{B}, 1 \to \mathsf{A}$**)** Suppose $w \in \Lambda_{\mathsf{B}}$, and assume the inductive hypothesis: $w \in X$ and 0, but not 00, is a suffix of w. We must show that $w1 \in X$ and 0 is not a suffix of $w1$. Because $w \in X$, we have that $w1 \in X$. And, 0 is not a suffix of $w1$.

Example Correctness Proof

- **(**$\mathsf{C}, 0 \to \mathsf{D}$**)** Suppose $w \in \Lambda_{\mathsf{C}}$, and assume the inductive hypothesis: $w \in X$ and 00 is a suffix of w. We must show that $w0 \notin X$. Because 00 is a suffix of w, we have that 000 is a suffix of $w0$. Thus $w0 \notin X$.
- **(**$\mathsf{C}, 1 \to \mathsf{A}$**)** Suppose $w \in \Lambda_{\mathsf{C}}$, and assume the inductive hypothesis: $w \in X$ and 00 is a suffix of w. We must show that $w1 \in X$ and 0 is not a suffix of $w1$. Because $w \in X$, we have that $w1 \in X$. And, 0 is not a suffix of $w1$.
- **(**$\mathsf{D}, 0 \to \mathsf{D}$**)** Suppose $w \in \Lambda_{\mathsf{D}}$, and assume the inductive hypothesis: $w \notin X$. We must show that $w0 \notin X$. Because $w \notin X$, we have that $w0 \notin X$.
- **(**$\mathsf{D}, 1 \to \mathsf{D}$**)** Suppose $w \in \Lambda_{\mathsf{D}}$, and assume the inductive hypothesis: $w \notin X$. We must show that $w1 \notin X$. Because $w \notin X$, we have that $w1 \notin X$.

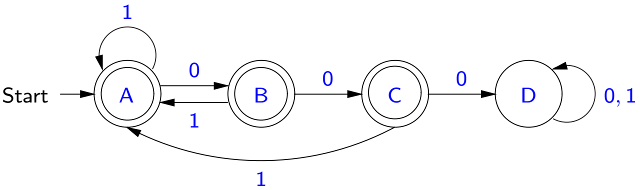
Example Correctness Proof

Now, we use the result of our induction on Λ to show that $L(M) = X$.

- **(**$L(M) \subseteq X$**)** Suppose $w \in L(M)$. Because $A_M = \{\text{A}, \text{B}, \text{C}\}$, we have that $w \in L(M) = \Lambda_\text{A} \cup \Lambda_\text{B} \cup \Lambda_\text{C}$. Thus, by parts (A)–(C), we have that $w \in X$.
- **(**$X \subseteq L(M)$**)** Suppose $w \in X$. Since $X \subseteq \{0, 1\}^*$, we have that $w \in \{0, 1\}^*$. Suppose, toward a contradiction, that $w \notin L(M)$. Because $w \notin L(M) = \Lambda_\text{A} \cup \Lambda_\text{B} \cup \Lambda_\text{C}$ and $w \in \{0, 1\}^* = (\textbf{alphabet}\, M)^* = \Lambda_\text{A} \cup \Lambda_\text{B} \cup \Lambda_\text{C} \cup \Lambda_\text{D}$, we must have that $w \in \Lambda_\text{D}$. But then part (D) tells us that $w \notin X$—contradiction. Thus $w \in L(M)$.

Simplification of DFAs

Let M be our example DFA



Is M simplified? No, since the state D is dead. But if we get rid of D, then we won't have a DFA anymore.

Thus, we will need:

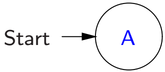
- a notion of when a DFA is simplified that is more liberal than our standard notion;
- a corresponding simplification procedure for DFAs.

Definition of Deterministically Simplified

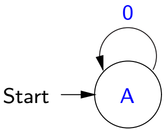
We say that a DFA M is *deterministically simplified* iff

- every element of Q_M is reachable; and
- at most one element of Q_M is dead.

For example, the following DFAs, which both accept $\emptyset$, are both deterministically simplified:



(M_1)



(M_2)

A Simplification Algorithm for DFAs

We define a simplification algorithm for DFAs that takes in

- a DFA M and
- an alphabet Σ

and returns a DFA N such that

- N is deterministically simplified,
- $N \approx M$,
- $\textbf{alphabet}\, N = \textbf{alphabet}(L(M)) \cup \Sigma$, and
- if $\Sigma \subseteq \textbf{alphabet}(L(M))$, then $|Q_N| \leq |Q_M|$.

A Simplification Algorithm

The algorithm begins by letting the FA M' be **simplify** M, i.e., the result of running our simplification algorithm for FAs on M. M' will have the following properties:

- $Q_{M'} \subseteq Q_M$ and $T_{M'} \subseteq T_M$;
- M' is simplified;
- $M' \approx M$;
- **alphabet** $M' =$ **alphabet**$(L(M')) =$ **alphabet**$(L(M))$; and
- for all $q \in Q_{M'}$ and $a \in$ **alphabet** M', there is at most one $r \in Q_{M'}$ such that $q, a \rightarrow r \in T_{M'}$ (this property holds since M is a DFA and $Q_{M'} \subseteq Q_M$ and $T_{M'} \subseteq T_M$).

A Simplification Algorithm

Let $\Sigma' = \textbf{alphabet}\, M' \cup \Sigma = \textbf{alphabet}(L(M)) \cup \Sigma$. If M' is a DFA and $\textbf{alphabet}\, M' = \Sigma'$, the algorithm returns M' as its DFA, N. Because M' is simplified, all states of M' are reachable, and either M' has no dead states, or it consists of a single dead state (the start state). In either case, M' is deterministically simplified. Because $Q_{M'} \subseteq Q_M$, we have $|Q_N| \leq |Q_M|$.

Otherwise, it must turn M' into a DFA whose alphabet is Σ'. We have that

- $\textbf{alphabet}\, M' \subseteq \Sigma'$; and
- for all $q \in Q_{M'}$ and $a \in \Sigma'$, there is at most one $r \in Q_{M'}$ such that $q, a \to r \in T_{M'}$.

Since M' is simplified, there are two cases to consider.

A Simplification Algorithm

If M' has no accepting states, then $s_{M'}$ is the only state of M' and M' has no transitions. Thus the DFA N returned by the algorithm is defined by:

- $Q_N = Q_{M'} = \{s_{M'}\}$;
- $s_N = s_{M'}$;
- $A_N = A_{M'} = \emptyset$; and
- $T_N = \{\, s_{M'}, a \to s_{M'} \mid a \in \Sigma' \,\}$.

In this case, we have that $|Q_N| \leq |Q_M|$.

A Simplification Algorithm

Alternatively, M' has at least one accepting state, so that M' has no dead states. (Consider the case when $\Sigma \subseteq \textbf{alphabet}(L(M))$, so that $\Sigma' = \textbf{alphabet}(L(M)) = \textbf{alphabet}\, M'$. Suppose, toward a contradiction, that $Q_{M'} = Q_M$, so that all elements of Q_M are useful. Then $s_{M'} = s_M$ and $A_{M'} = A_M$. And $T_{M'} = T_M$, since no transitions of a DFA are redundant. Hence $M' = M$, so that M' is a DFA with alphabet Σ'—a contradiction. Thus $Q_{M'} \subsetneq Q_M$.)

A Simplification Algorithm

Thus the DFA N returned by the algorithm is defined by:

- $Q_N = Q_{M'} \cup \{\langle\text{dead}\rangle\}$ (enough brackets are put around $\langle\text{dead}\rangle$ so that it's not in $Q_{M'}$);
- $s_N = s_{M'}$;
- $A_N = A_{M'}$; and
- $T_N = T_{M'} \cup T'$, where T' is the set of all transitions $q, a \to \langle\text{dead}\rangle$ such that either
  - $q \in Q_{M'}$ and $a \in \Sigma'$, but there is no $r \in Q_{M'}$ such that $q, a \to r \in T_{M'}$; or
  - $q = \langle\text{dead}\rangle$ and $a \in \Sigma'$.

(If $\Sigma \subseteq \textbf{alphabet}(L(M))$, then $|Q_N| \leq |Q_M|$.)

Definition of **determSimplify** *Function*

We define a function **determSimplify** $\in$ **DFA** $\times$ **Alp** $\rightarrow$ **DFA** by: **determSimplify**(M, Σ) is the result of running the above algorithm on M and Σ.

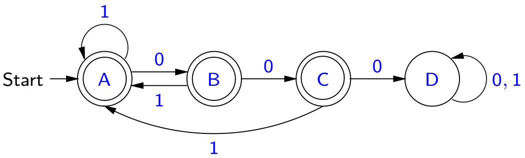
Theorem 3.11.8

For all $M \in$ **DFA** *and* $\Sigma \in$ **Alp***:*

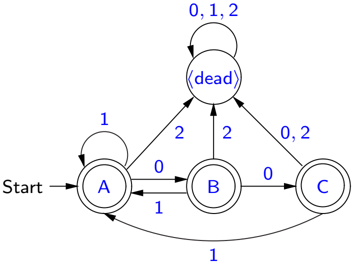
- **determSimplify**(M, Σ) *is deterministically simplified;*
- **determSimplify**$(M, \Sigma) \approx M$*;*
- **alphabet**(**determSimplify**(M, Σ)) $=$ **alphabet**$(L(M)) \cup \Sigma$*; and*
- *if* $\Sigma \subseteq$ **alphabet**$(L(M))$*, then* $|Q_{\mathbf{determSimplify}(M,\Sigma)}| \leq |Q_M|$*.*

Example DFA Simplification

For example, suppose M is the DFA

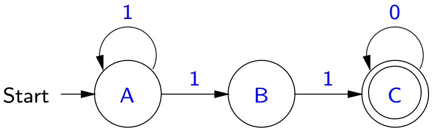


Then **determSimplify**$(M, \{2\})$ is the DFA



Converting NFAs to DFAs

Suppose M is the NFA



How can we convert M into a DFA?

Our approach will be to convert M into a DFA N whose states represent the elements of the set

$$\{\, \Delta_M(\{A\}, w) \mid w \in \{0,1\}^* \,\}.$$

For example, one the states of N will be $\langle A, B \rangle$, which represents $\{A, B\} = \Delta_M(\{A\}, 1)$. This is the state that our DFA will be in after processing 1 from the start state.

A Proposition About Δ for NFAs

Proposition 3.11.10

Suppose M is an NFA.

(1) *For all $P \subseteq Q_M$, $\Delta_M(P, \%) = P$.*

(2) *For all $P \subseteq Q_M$ and $a \in \textbf{alphabet}\, M$,*
$\Delta_M(P, a) = \{\, r \in Q_M \mid p, a \to r \in T_M, \text{ for some } p \in P \,\}$.

(3) *For all $P \subseteq Q_M$ and $x, y \in (\textbf{alphabet}\, M)^*$,*
$\Delta_M(P, xy) = \Delta_M(\Delta_M(P, x), y)$.

Representing Finite Sets of Symbols as Symbols

Given a finite set of symbols P, we write $\overline{P}$ for the symbol

$$\langle a_1, \ldots, a_n \rangle,$$

where $a_1, \ldots, a_n$ are all of the elements of P, in order according to our ordering on **Sym**, and without repetition. For example, $\overline{\{\mathsf{B}, \mathsf{A}\}} = \langle \mathsf{A}, \mathsf{B} \rangle$ and $\overline{\emptyset} = \langle \rangle$.

It is easy to see that, if P and R are finite sets of symbols, then $\overline{P} = \overline{R}$ iff $P = R$.

Our NFA to DFA Conversion Algorithm

We convert an NFA M into a DFA N as follows. First, we generate the least subset X of $\mathcal{P}\, Q_M$ such that:

- $\{s_M\} \in X$;
- for all $P \in X$ and $a \in \textbf{alphabet}\, M$, $\Delta_M(P, a) \in X$.

Thus $|X| \leq 2^{|Q_M|}$.

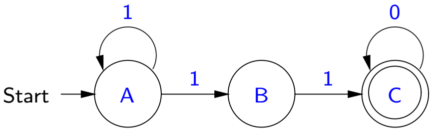
Then we define the DFA N as follows:

- $Q_N = \{\, \overline{P} \mid P \in X \,\}$;
- $s_N = \overline{\{s_M\}} = \langle s_M \rangle$;
- $A_N = \{\, \overline{P} \mid P \in X$ and $P \cap A_M \neq \emptyset \,\}$;
- $T_N = \{\, (\overline{P}, a, \overline{\Delta_M(P, a)}) \mid P \in X$ and $a \in \textbf{alphabet}\, M \,\}$.

Then N is a DFA with alphabet $\textbf{alphabet}\, M$ and, for all $P \in X$ and $a \in \textbf{alphabet}\, M$, $\delta_N(\overline{P}, a) = \overline{\Delta_M(P, a)}$.

Conversion Example

Suppose M is the NFA



Let's work out what the DFA N is.

- To begin with, $\{\text{A}\} \in X$, so that $\langle\text{A}\rangle \in Q_N$. And $\langle\text{A}\rangle$ is the start state of N. It is not an accepting state, since $\text{A} \notin A_M$.
- Since $\{\text{A}\} \in X$, and $\Delta(\{\text{A}\}, 0) = \emptyset$, we add $\emptyset$ to X, $\langle\rangle$ to Q_N and $\langle\text{A}\rangle, 0 \to \langle\rangle$ to T_N.

  Since $\{\text{A}\} \in X$, and $\Delta(\{\text{A}\}, 1) = \{\text{A}, \text{B}\}$, we add $\{\text{A}, \text{B}\}$ to X, $\langle\text{A}, \text{B}\rangle$ to Q_N and $\langle\text{A}\rangle, 1 \to \langle\text{A}, \text{B}\rangle$ to T_N.

Conversion Example

- Since $\emptyset \in X$, $\Delta(\emptyset, 0) = \emptyset$ and $\emptyset \in X$, we don't have to add anything to X or Q_N, but we add $\langle\rangle, 0 \rightarrow \langle\rangle$ to T_N.

  Since $\emptyset \in X$, $\Delta(\emptyset, 1) = \emptyset$ and $\emptyset \in X$, we don't have to add anything to X or Q_N, but we add $\langle\rangle, 1 \rightarrow \langle\rangle$ to T_N.

- Since $\{A, B\} \in X$, $\Delta(\{A, B\}, 0) = \emptyset$ and $\emptyset \in X$, we don't have to add anything to X or Q_N, but we add $\langle A, B\rangle, 0 \rightarrow \langle\rangle$ to T_N.

  Since $\{A, B\} \in X$, $\Delta(\{A, B\}, 1) = \{A, B\} \cup \{C\} = \{A, B, C\}$, we add $\{A, B, C\}$ to X, $\langle A, B, C\rangle$ to Q_N, and $\langle A, B\rangle, 1 \rightarrow \langle A, B, C\rangle$ to T_N. Since $\{A, B, C\}$ contains (the only) one of M's accepting states, we add $\langle A, B, C\rangle$ to A_N.

Conversion Example

- Since $\{A, B, C\} \in X$ and $\Delta(\{A, B, C\}, 0) = \emptyset \cup \emptyset \cup \{C\} = \{C\}$, we add $\{C\}$ to X, $\langle C \rangle$ to Q_N and $\langle A, B, C \rangle, 0 \rightarrow \langle C \rangle$ to T_N. Since $\{C\}$ contains one of M's accepting states, we add $\langle C \rangle$ to A_N.

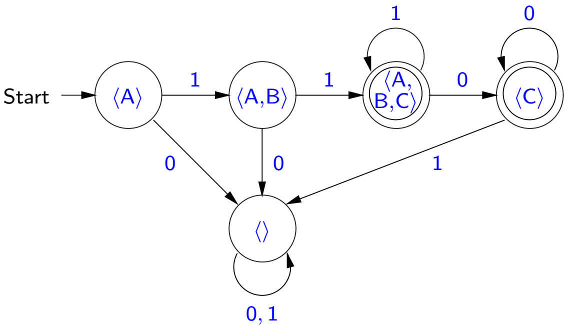
  Since $\{A, B, C\} \in X$, $\Delta(\{A, B, C\}, 1) = \{A, B\} \cup \{C\} \cup \emptyset = \{A, B, C\}$ and $\{A, B, C\} \in X$, we don't have to add anything to X or Q_N, but we add $\langle A, B, C \rangle, 1 \rightarrow \langle A, B, C \rangle$ to T_N.

- Since $\{C\} \in X$, $\Delta(\{C\}, 0) = \{C\}$ and $\{C\} \in X$, we don't have to add anything to X or Q_N, but we add $\langle C \rangle, 0 \rightarrow \langle C \rangle$ to T_N.

  Since $\{C\} \in X$, $\Delta(\{C\}, 1) = \emptyset$ and $\emptyset \in X$, we don't have to add anything to X or Q_N, but we add $\langle C \rangle, 1 \rightarrow \langle \rangle$ to T_N.

Conversion Example

Since there are no more elements to add to X, we are done. Thus, the DFA N is



Correctness of Conversion Algorithm

Lemma 3.11.11

For all $w \in (\textbf{alphabet}\ M)^*$*:*

- $\Delta_M(\{s_M\}, w) \in X$*; and*
- $\delta_N(s_N, w) = \overline{\Delta_M(\{s_M\}, w)}$.

Proof. By left string induction.

(Basis Step) We have that $\Delta_M(\{s_M\}, \%) = \{s_M\} \in X$ and $\delta_N(s_N, \%) = s_N = \overline{\{s_M\}} = \overline{\Delta_M(\{s_M\}, \%)}$.

Correctness

Proof (cont.). (Inductive Step) Suppose $a \in \textbf{alphabet}\, M$ and $w \in (\textbf{alphabet}\, M)^*$. Assume the inductive hypothesis: $\Delta_M(\{s_M\}, w) \in X$ and $\delta_N(s_N, w) = \overline{\Delta_M(\{s_M\}, w)}$.

Since $\Delta_M(\{s_M\}, w) \in X$ and $a \in \textbf{alphabet}\, M$, we have that $\Delta_M(\{s_M\}, wa) = \Delta_M(\Delta_M(\{s_M\}, w), a) \in X$. Thus

$$\begin{aligned}
\delta_N(s_N, wa) &= \delta_N(\delta_N(s_N, w), a) \\
&= \delta_N(\overline{\Delta_M(\{s_M\}, w)}, a) \qquad \text{(ind. hyp.)} \\
&= \overline{\Delta_M(\Delta_M(\{s_M\}, w), a)} \\
&= \overline{\Delta_M(\{s_M\}, wa)}.
\end{aligned}$$

□

Lemma 3.11.12
$L(N) = L(M)$.

Proof. $(L(M) \subseteq L(N))$ Suppose $w \in L(M)$, so that $w \in (\textbf{alphabet}\ M)^* = (\textbf{alphabet}\ N)^*$ and $\Delta_M(\{s_M\}, w) \cap A_M \neq \emptyset$. By Lemma 3.11.11, we have that $\Delta_M(\{s_M\}, w) \in X$ and $\delta_N(s_N, w) = \Delta_M(\{s_M\}, w)$. Since $\Delta_M(\{s_M\}, w) \in X$ and $\Delta_M(\{s_M\}, w) \cap A_M \neq \emptyset$, it follows that $\delta_N(s_N, w) = \Delta_M(\{s_M\}, w) \in A_N$. Thus $w \in L(N)$.

$(L(N) \subseteq L(M))$ Suppose $w \in L(N)$, so that $w \in (\textbf{alphabet}\ N)^* = (\textbf{alphabet}\ M)^*$ and $\delta_N(s_N, w) \in A_N$. By Lemma 3.11.11, we have that $\delta_N(s_N, w) = \Delta_M(\{s_M\}, w)$. Thus $\Delta_M(\{s_M\}, w) \in A_N$, so that $\Delta_M(\{s_M\}, w) \cap A_M \neq \emptyset$. Thus $w \in L(M)$. □

Conversion Function

We define a function **nfaToDFA** $\in$ **NFA** $\to$ **DFA** by: **nfaToDFA** M is the result of running the preceding algorithm with input M.

Theorem 3.11.13
For all $M \in$ **NFA***:*

- **nfaToDFA** $M \approx M$*; and*
- **alphabet(nfaToDFA** M**)** $=$ **alphabet** M*.*

Processing DFAs in Forlan

The Forlan module `DFA` defines an abstract type `dfa` (in the top-level environment) of deterministic finite automata, along with various functions for processing DFAs.

Values of type `dfa` are implemented as values of type `fa`, and the module DFA provides the following injection and projection functions

```
val injToFA     : dfa -> fa
val injToEFA    : dfa -> efa
val injToNFA    : dfa -> nfa
val projFromFA  : fa -> dfa
val projFromEFA : efa -> dfa
val projFromNFA : nfa -> dfa
```

These functions are available in the top-level environment with the names `injDFAToFA`, `injDFAToEFA`, `injDFAToNFA`, `projFAToDFA`, `projEFAToDFA` and `projNFAToDFA`.

Processing DFAs in Forlan

The module DFA also defines the functions:

```
val input             : string -> dfa
val determProcStr     : dfa -> sym * str -> sym
val determAccepted    : dfa -> str -> bool
val determSimplified : dfa -> bool
val determSimplify    : dfa * sym set -> dfa
val fromNFA           : nfa -> dfa
```

The last of these functions is available in the top-level environment as:

```
val nfaToDFA : nfa -> dfa
```

Processing DFAs in Forlan

Most of the functions for processing FAs that were introduced in previous sections are inherited by `DFA`:

```
val output                    : string * dfa -> unit
val numStates                 : dfa -> int
val numTransitions            : dfa -> int
val alphabet                  : dfa -> sym set
val equal                     : dfa * dfa -> bool
val checkLP                   : dfa -> lp -> unit
val validLP                   : dfa -> lp -> bool
val isomorphism               : dfa * dfa * sym_rel -> bool
val findIsomorphism           : dfa * dfa -> sym_rel
val isomorphic                : dfa * dfa -> bool
val renameStates              : dfa * sym_rel -> dfa
val renameStatesCanonically : dfa -> dfa
```

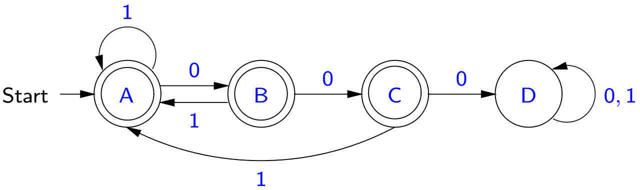
Processing DFAs in Forlan

More inherited functions:

```
val processStr       : dfa -> sym set * str -> sym set
val accepted         : dfa -> str -> bool
val findLP           : dfa -> sym set * str * sym set -> lp
val findAcceptingLP : dfa -> str -> lp
```

Forlan Examples

Suppose `dfa` is the `dfa`



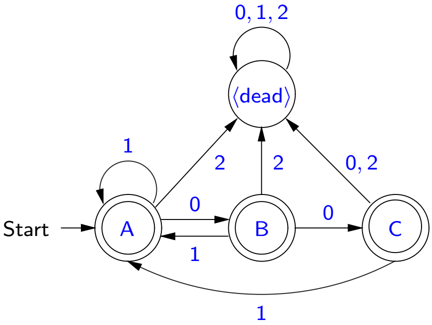
We can turn `dfa` into an equivalent deterministically simplified DFA whose alphabet is the union of the alphabet of the language of `dfa` and $\{2\}$, i.e., whose alphabet is $\{0, 1, 2\}$, as follows.

```
- val dfa' = DFA.determSimplify(dfa, SymSet.input "");
@ 2
@ .
val dfa' = - : dfa
```

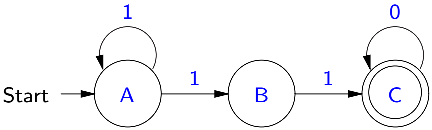
```
- DFA.output("", dfa');
{states} A, B, C, <dead> {start state} A
{accepting states} A, B, C
{transitions}
A, 0 -> B; A, 1 -> A; A, 2 -> <dead>; B, 0 -> C;
B, 1 -> A; B, 2 -> <dead>; C, 0 -> <dead>; C, 1 -> A;
C, 2 -> <dead>; <dead>, 0 -> <dead>;
<dead>, 1 -> <dead>; <dead>, 2 -> <dead>
val it = () : unit
```

Forlan Examples

Thus `dfa'` is



Suppose that `nfa` is the `nfa`

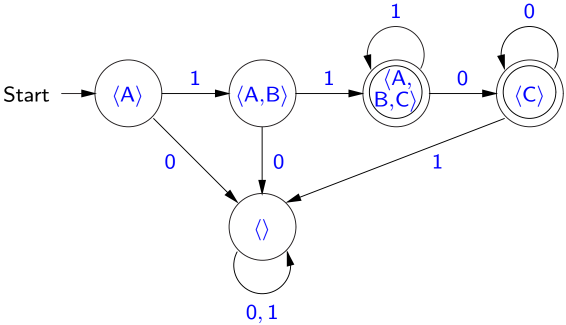


We can convert `nfa` to a DFA as follows:

```
- val dfa = nfaToDFA nfa;
val dfa = - : dfa
```

```
- DFA.output("", dfa);
{states} <>, <A>, <C>, <A,B>, <A,B,C>
{start state} <A> {accepting states} <C>, <A,B,C>
{transitions}
<>, 0 -> <>; <>, 1 -> <>; <A>, 0 -> <>;
<A>, 1 -> <A,B>; <C>, 0 -> <C>; <C>, 1 -> <>;
<A,B>, 0 -> <>; <A,B>, 1 -> <A,B,C>;
<A,B,C>, 0 -> <C>; <A,B,C>, 1 -> <A,B,C>
val it = () : unit
```

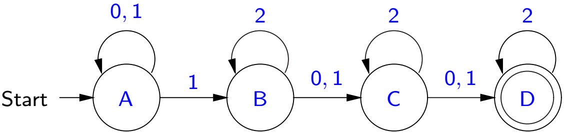
Thus dfa is



Forlan Examples

Finally, we see an example in which an NFA with 4 states is converted to a DFA with $2^4 = 16$ states.

Suppose `nfa'` is the NFA



We can convert `nfa'` into a DFA, as follows:

```
- val dfa' = nfaToDFA nfa';
val dfa' = - : dfa
- DFA.numStates dfa';
val it = 16 : int
```

In Section 3.13, we will use Forlan to show that there is no DFA with fewer than 16 states that accepts the language accepted by `nfa'` and `dfa'`.