3.13: Equivalence-testing and Minimization of Deterministic Finite Automata

In this section, we give algorithms for:

- testing whether two DFAs are equivalent; and
- minimizing the alphabet size and number of states of a DFA.

We also show how to use the Forlan implementations of these algorithms.

Testing the Equivalence of DFAs

Suppose M and N are DFAs. Our algorithm for checking whether they are equivalent proceeds as follows.

First, it converts M and N into DFAs with identical alphabets. Let $\Sigma = \textbf{alphabet}\, M \cup \textbf{alphabet}\, N$, and define the DFAs M' and N' by:

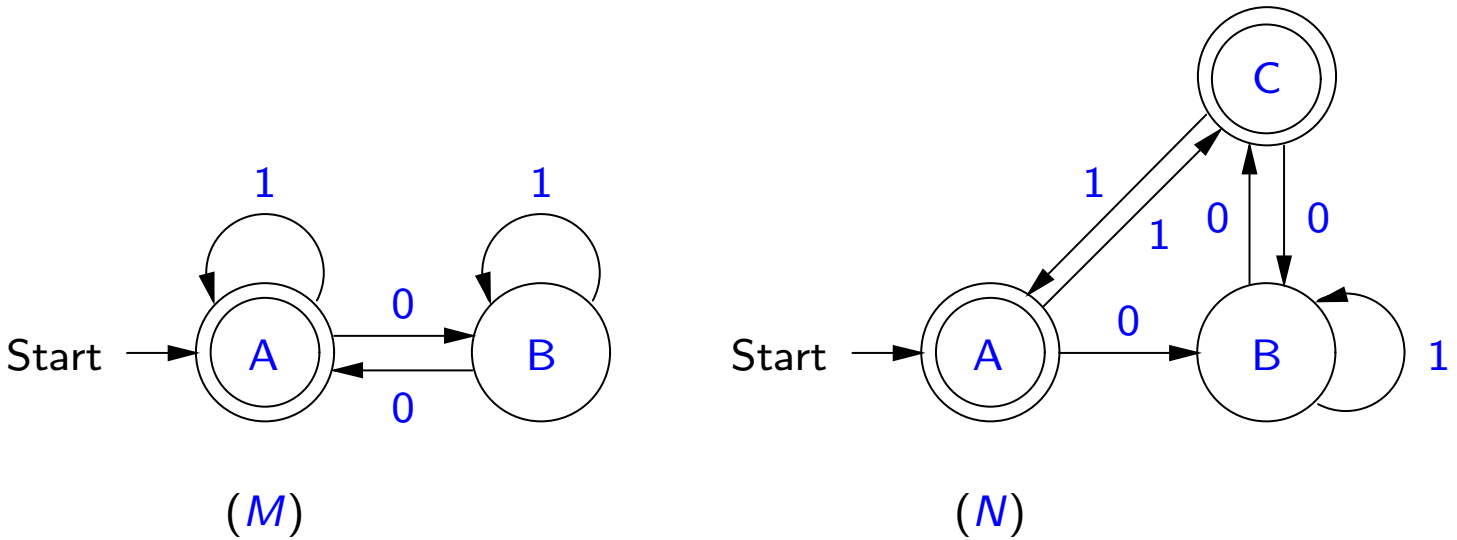
$$M' = \textbf{determSimplify}(M, \Sigma), \text{ and}$$
$$N' = \textbf{determSimplify}(N, \Sigma).$$

Since $\textbf{alphabet}(L(M)) \subseteq \textbf{alphabet}\, M \subseteq \Sigma$, we have that $\textbf{alphabet}\, M' = \textbf{alphabet}(L(M)) \cup \Sigma = \Sigma$. Similarly, $\textbf{alphabet}\, N' = \Sigma$.

Furthermore, $M' \approx M$ and $N' \approx N$, so that it will suffice to determine whether M' and N' are equivalent.

Equivalence-testing

For example, if M and N are the DFAs



(M) (N)

then $\Sigma = \{0, 1\}$, $M' = M$ and $N' = N$.

Equivalence-testing

Next, the algorithm generates the least subset X of $Q_{M'} \times Q_{N'}$ such that

- $(s_{M'}, s_{N'}) \in X$; and
- for all $q \in Q_{M'}$, $r \in Q_{N'}$ and $a \in \Sigma$, if $(q, r) \in X$, then $(\delta_{M'}(q, a), \delta_{N'}(r, a)) \in X$.

With our example DFAs M' and N', we have that

- $(\mathrm{A}, \mathrm{A}) \in X$;
- since $(\mathrm{A}, \mathrm{A}) \in X$, we have that $(\mathrm{B}, \mathrm{B}) \in X$ and $(\mathrm{A}, \mathrm{C}) \in X$;
- since $(\mathrm{B}, \mathrm{B}) \in X$, we have that (again) $(\mathrm{A}, \mathrm{C}) \in X$ and (again) $(\mathrm{B}, \mathrm{B}) \in X$; and
- since $(\mathrm{A}, \mathrm{C}) \in X$, we have that (again) $(\mathrm{B}, \mathrm{B}) \in X$ and (again) $(\mathrm{A}, \mathrm{A}) \in X$.

Equivalence-testing

Back in the general case, we have the following lemmas.

Lemma 3.13.1
For all $w \in \Sigma^*$, $(\delta_{M'}(s_{M'}, w), \delta_{N'}(s_{N'}, w)) \in X$.

Proof. By left string induction on w. □

Lemma 3.13.2
For all $q \in Q_{M'}$ *and* $r \in Q_{N'}$, *if* $(q, r) \in X$, *then there is a* $w \in \Sigma^*$ *such that* $q = \delta_{M'}(s_{M'}, w)$ *and* $r = \delta_{N'}(s_{N'}, w)$.

Proof. By induction on X. □

Finally, the algorithm checks that, for all $(q, r) \in X$,

$$q \in A_{M'} \quad \text{iff} \quad r \in A_{N'}.$$

If this is true, it says that the machines are equivalent; otherwise it says they are not equivalent.

Equivalence-testing

Suppose every pair $(q, r) \in X$ consists of two accepting states or of two non-accepting states. Suppose, toward a contradiction, that $L(M') \neq L(N')$. Then there is a string w that is accepted by one of the machines but is not accepted by the other. Since both machines have alphabet Σ, Lemma 3.13.1 tells us that $(\delta_{M'}(s_{M'}, w), \delta_{N'}(s_{N'}, w)) \in X$. But one side of this pair is an accepting state and the other is a non-accepting one—contradiction. Thus $L(M') = L(N')$.

Suppose we find a pair $(q, r) \in X$ such that one of q and r is an accepting state but the other is not. By Lemma 3.13.2, it will follow that there is a string w that is accepted by one of the machines but not accepted by the other one, i.e., that $L(M') \neq L(N')$.

Equivalence-testing

In the case of our example, we have that $X = \{(A, A), (B, B), (A, C)\}$. Since (A, A) and (A, C) are pairs of accepting states, and (B, B) is a pair of non-accepting states, it follows that $L(M') = L(N')$. Hence $L(M) = L(N)$.

By annotating each element $(q, r) \in X$ with a string w such that $q = \delta_{M'}(s_{M'}, w)$ and $r = \delta_{N'}(s_{N'}, w)$, instead of just reporting that M' and N' are not equivalent, we can explain why they are not equivalent,

- giving a string that is accepted by the first machine but not by the second; and/or
- giving a string that is accepted by the second machine but not by the first.

We can even arrange for these strings to be of minimum length. The Forlan implementation of our algorithm always produces minimum-length counterexamples.

Equivalence-testing in Forlan

The Forlan module `DFA` defines the functions:

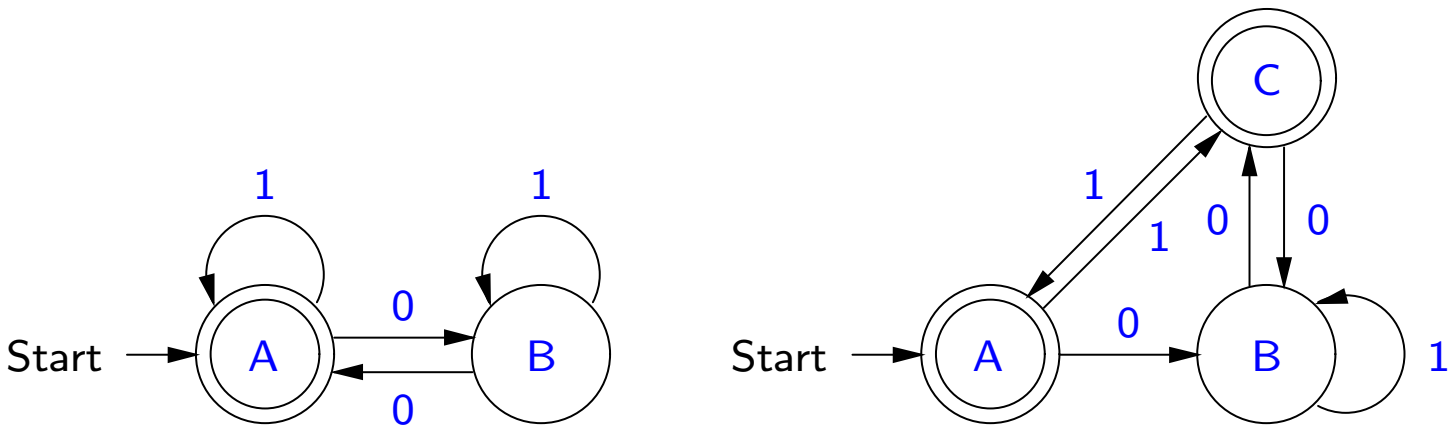
```
val relationship : dfa * dfa -> unit
val subset       : dfa * dfa -> bool
val equivalent   : dfa * dfa -> bool
```

The function `relationship` figures out the relationship between the languages accepted by two DFAs (are they equal, is one a proper subset of the other, is neither a subset of the other), and supplies minimum-length counterexamples to justify negative answers. The function `subset` tests whether its first argument's language is a subset of its second argument's language. The function `equivalent` tests whether two DFAs are equivalent.

Note that `subset` (when turned into a function of type `reg * reg -> bool`—see below) can be used in conjunction with the local and global simplification functions of Section 3.3.

Forlan Examples

For example, suppose `dfa1` and `dfa2` of type `dfa` are bound to our example DFAs M and N, respectively:

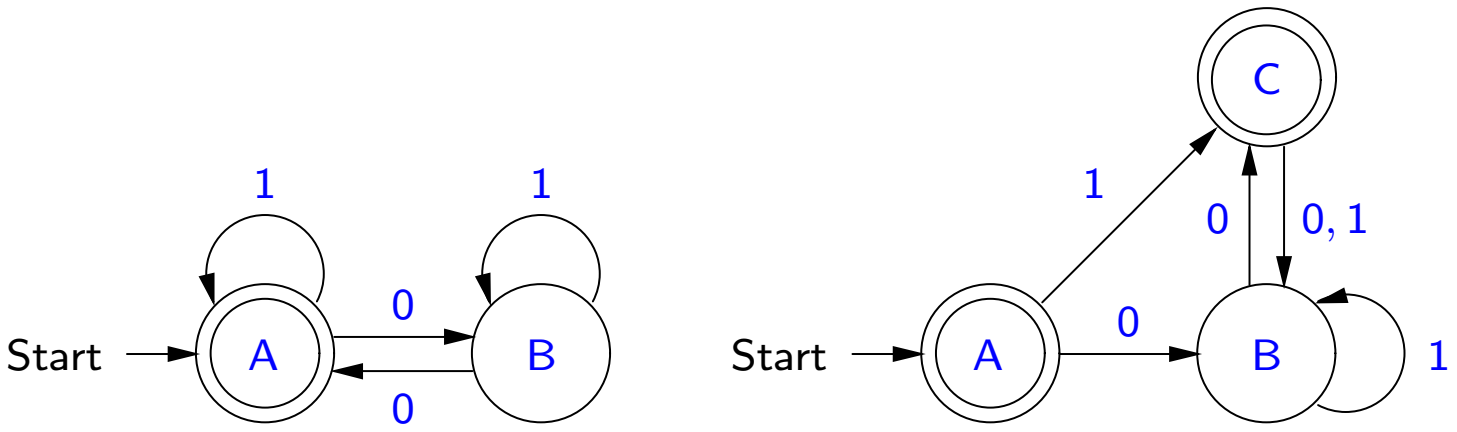


We can verify that these machines are equivalent as follows:

```
- DFA.relationship(dfa1, dfa2);
languages are equal
val it = () : unit
```

Forlan Examples

On the other hand, suppose that `dfa3` and `dfa4` of type `dfa` are bound to the DFAs:



We can find out why these machines are not equivalent as follows:

```
- DFA.relationship(dfa3, dfa4);
neither language is a subset of the other language:
"11" is in first language but is not in second
language; "110" is in second language but is not in
first language
val it = () : unit
```

Forlan Examples

We can find the relationship between the languages generated by regular expressions `reg1` and `reg2` by:

- converting `reg1` and `reg2` to DFAs `dfa1` and `dfa2`, and then
- running `DFA.relationship(dfa1, dfa2)` to find the relationship between those DFAs.

Of course, we can define an ML/Forlan function that carries out these actions:

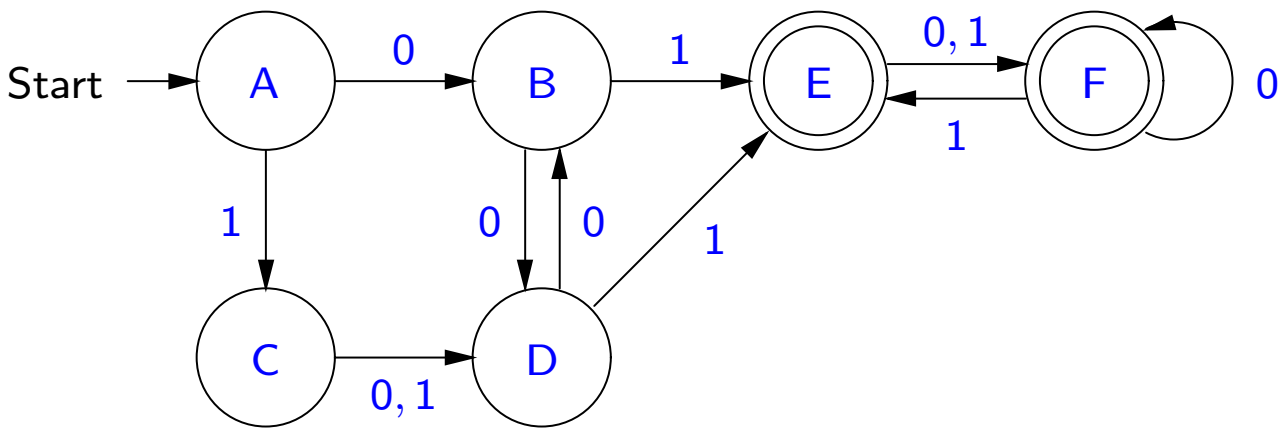
```
- fun regToDFA reg =
=       nfaToDFA(efaToNFA(faToEFA(regToFA reg)));
val regToDFA = fn : reg -> dfa
- fun relationshipReg(reg1, reg2) =
=       DFA.relationship
=       (regToDFA reg1, regToDFA reg2);
val relationshipReg = fn : reg * reg -> unit
```

Minimization of DFAs

Now, we consider an algorithm for minimizing the sizes of the alphabet and set of states of a DFA M.

The algorithm first minimizes the size of M's alphabet, and makes the automaton be deterministically simplified, by letting $M' = \textbf{determSimplify}(M, \emptyset)$. Thus $M' \approx M$ and $\textbf{alphabet}\ M' = \textbf{alphabet}(L(M))$.

For example, if M is the DFA



then $M' = M$.

Unmergable States

Next, the algorithm generates the least subset X of $Q_{M'} \times Q_{M'}$ such that:

(1) $A_{M'} \times (Q_{M'} - A_{M'}) \subseteq X$;

(2) $(Q_{M'} - A_{M'}) \times A_{M'} \subseteq X$; and

(3) for all $q, q', r, r' \in Q_{M'}$ and $a \in$ **alphabet** M', if $(q, r) \in X$, $(q', a, q) \in T_{M'}$ and $(r', a, r) \in T_{M'}$, then $(q', r') \in X$.

We read "$(q, r) \in X$" as "q and r cannot be merged".

Unmergable States Example

In the case of our example M', (1) tells us to add the pairs (E, A), (E, B), (E, C), (E, D), (F, A), (F, B), (F, C) and (F, D) to X.

And, (2) tells us to add the pairs (A, E), (B, E), (C, E), (D, E), (A, F), (B, F), (C, F) and (D, F) to X.

Now we use rule (3) to compute the rest of X's elements. To begin with, we must handle each pair that has already been added to X.

- Since there are no transitions leading into A, no pairs can be added using (E, A), (A, E), (F, A) and (A, F).
- Since there are no 0-transitions leading into E, and there are no 1-transitions leading into B, no pairs can be added using (E, B) and (B, E).

Unmergable States Example

- Since $(E, C), (C, E) \in X$ and $(B, 1, E)$, $(D, 1, E)$, $(F, 1, E)$ and $(A, 1, C)$ are the 1-transitions leading into E and C, we add (B, A) and (A, B), and (D, A) and (A, D) to X; we would also have added (F, A) and (A, F) to X if they hadn't been previously added. Since there are no 0-transitions into E, nothing can be added to X using (E, C) and (C, E) and 0-transitions.
- Since $(E, D), (D, E) \in X$ and $(B, 1, E)$, $(D, 1, E)$, $(F, 1, E)$ and $(C, 1, D)$ are the 1-transitions leading into E and D, we add (B, C) and (C, B), and (D, C) and (C, D) to X; we would also have added (F, C) and (C, F) to X if they hadn't been previously added. Since there are no 0-transitions into E, nothing can be added to X using (E, D) and (D, E) and 0-transitions.

Unmergable States Example

- Since $(F, B), (B, F) \in X$ and $(E, 0, F)$, $(F, 0, F)$, $(A, 0, B)$, and $(D, 0, B)$ are the 0-transitions leading into F and B, we would have to add the following pairs to X, if they were not already present: (E, A), (A, E), (E, D), (D, E), (F, A), (A, F), (F, D), (D, F). Since there are no 1-transitions leading into B, no pairs can be added using (F, B) and (B, F) and 1-transitions.
- Since $(F, C), (C, F) \in X$ and $(E, 1, F)$ and $(A, 1, C)$ are the 1-transitions leading into F and C, we would have to add (E, A) and (A, E) to X if these pairs weren't already present. Since there are no 0-transitions leading into C, no pairs can be added using (F, C) and (C, F) and 0-transitions.

Unmergable States Example

- Since $(F, D), (D, F) \in X$ and $(E, 0, F)$, $(F, 0, F)$, $(B, 0, D)$ and $(C, 0, D)$ are the 0-transitions leading into F and D, we would add (E, B), (B, E), (E, C), (C, E), (F, B), (B, F), (F, C), and (C, F) to X, if these pairs weren't already present. Since $(F, D), (D, F) \in X$ and $(E, 1, F)$ and $(C, 1, D)$ are the 1-transitions leading into F and D, we would add (E, C) and (C, E) to X, if these pairs weren't already in X.

Unmergable States Example

We've now handled all of the elements of X that were added using rules (1) and (2). We must now handle the pairs that were subsequently added: (A, B), (B, A), (A, D), (D, A), (B, C), (C, B), (C, D), (D, C).

- Since there are no transitions leading into A, no pairs can be added using (A, B), (B, A), (A, D) and (D, A).
- Since there are no 1-transitions leading into B, and there are no 0-transitions leading into C, no pairs can be added using (B, C) and (C, B).
- Since $(C, D), (D, C) \in X$ and $(A, 1, C)$ and $(C, 1, D)$ are the 1-transitions leading into C and D, we add the pairs (A, C) and (C, A) to X. Since there are no 0-transitions leading into C, no pairs can be added to X using (C, D) and (D, C) and 0-transitions.

Unmergable States Example

Now, we must handle the pairs that were added in the last phase: (A, C) and (C, A).

- Since there are no transitions leading into A, no pairs can be added using (A, C) and (C, A).

Since we have handled all the pairs we added to X, we are now done. Here are the 26 elements of X: (A, B), (A, C), (A, D), (A, E), (A, F), (B, A), (B, C), (B, E), (B, F), (C, A), (C, B), (C, D), (C, E), (C, F), (D, A), (D, C), (D, E), (D, F), (E, A), (E, B), (E, C), (E, D), (F, A), (F, B), (F, C), (F, D).

Unmergable States Lemmas

Back in the general case, we have the following lemmas.

Lemma 3.13.3
For all $(q, r) \in X$, there is a $w \in (\textbf{alphabet}\ M')^*$, such that exactly one of $\delta_{M'}(q, w)$ and $\delta_{M'}(r, w)$ is in $A_{M'}$.

Proof. By induction on X. □

Lemma 3.13.4
For all $w \in (\textbf{alphabet}\ M')^*$, for all $q, r \in Q_{M'}$, if exactly one of $\delta_{M'}(q, w)$ and $\delta_{M'}(r, w)$ is in $A_{M'}$, then $(q, r) \in X$.

Proof. By right string induction. □

Mergable States

The algorithm now lets the relation $Y = (Q_{M'} \times Q_{M'}) - X$. We read "$(q, r) \in Y$" as "$q$ and r can be merged".

Back with our example, we have that Y is

$$\{(A, A), (B, B), (C, C), (D, D), (E, E), (F, F)\}$$
$$\cup$$
$$\{(B, D), (D, B), (F, E), (E, F)\}.$$

Mergeable States Lemmas

Lemma 3.13.5

(1) *For all* $q, r \in Q_{M'}$, $(q, r) \in Y$ *iff, for all* $w \in (\textbf{alphabet}\ M')^*$, $\delta_{M'}(q, w) \in A_{M'}$ *iff* $\delta_{M'}(r, w) \in A_{M'}$.

(2) *For all* $q, r \in Q_{M'}$, *if* $(q, r) \in Y$, *then* $q \in A_{M'}$ *iff* $r \in A_{M'}$.

(3) *For all* $q, r \in Q_{M'}$ *and* $a \in \textbf{alphabet}\ M'$, *if* $(q, r) \in Y$, *then* $(\delta_{M'}(q, a), \delta_{M'}(r, a)) \in Y$.

Proof.

(1) Follows using Lemmas 3.13.3 and 3.13.4.

(2) Follows by Part (1), when $w = \%$.

(3) Follows by Part (1).

□

Mergable States Lemmas

The following lemma says that Y is an *equivalence relation on* $Q_{M'}$.

Lemma 3.13.6

Y is reflexive on $Q_{M'}$, symmetric and transitive.

Proof. Follows from Lemma 3.13.5(1). □

Equivalence Classes

In order to define the DFA N that is the result of our minimization algorithm, we need a bit more notation.

As in Section 3.11, we write $\overline{P}$ for the result of coding a finite set of symbols P as a symbol. E.g., $\overline{\{\mathrm{B}, \mathrm{A}\}} = \langle \mathrm{A}, \mathrm{B} \rangle$.

If $q \in Q_{M'}$, we write $[q]$ for $\{\, p \in Q_{M'} \mid (p, q) \in Y \,\}$, which is called the *equivalence class* of q. Using Lemma 3.13.6, it is easy to show that, $q \in [q]$, for all $q \in Q_{M'}$, and $[q] = [r]$ iff $(q, r) \in Y$, for all $q, r \in Q_{M'}$.

If P is a nonempty, finite set of symbols, then we write $\min P$ for the least element of P, according to our standard ordering on symbols.

Definition of Minimized DFA

Next, the algorithm lets $Z = \{\, [q] \mid q \in Q_{M'} \,\}$, which is finite since $Q_{M'}$ is finite.

In the case of our example, Z is

$$\{\{\mathsf{A}\}, \{\mathsf{B}, \mathsf{D}\}, \{\mathsf{C}\}, \{\mathsf{E}, \mathsf{F}\}\}.$$

The algorithm then defines the DFA N as follows:

- $Q_N = \{\, \overline{P} \mid P \in Z \,\}$;
- $s_N = \overline{[s_{M'}]}$;
- $A_N = \{\, \overline{P} \mid P \in Z \text{ and } \min P \in A_{M'} \,\}$; and
- $T_N =$
  $\{\, (\overline{P}, a, \overline{[\delta_{M'}(\min P, a)]}) \mid P \in Z \text{ and } a \in \textbf{alphabet}\, M' \,\}$.

Then N is a DFA with alphabet **alphabet** M' and, for all $P \in Z$ and $a \in$ **alphabet** M', $\delta_N(\overline{P}, a) = \overline{[\delta_{M'}(\min P, a)]}$.

Example Minimization

In the case of our example, we have that

- $Q_N = \{\langle \mathrm{A} \rangle, \langle \mathrm{B}, \mathrm{D} \rangle, \langle \mathrm{C} \rangle, \langle \mathrm{E}, \mathrm{F} \rangle\}$;
- $s_N = \langle A \rangle$; and
- $A_N = \{\langle \mathrm{E}, \mathrm{F} \rangle\}$.

We compute the elements of T_N as follows.

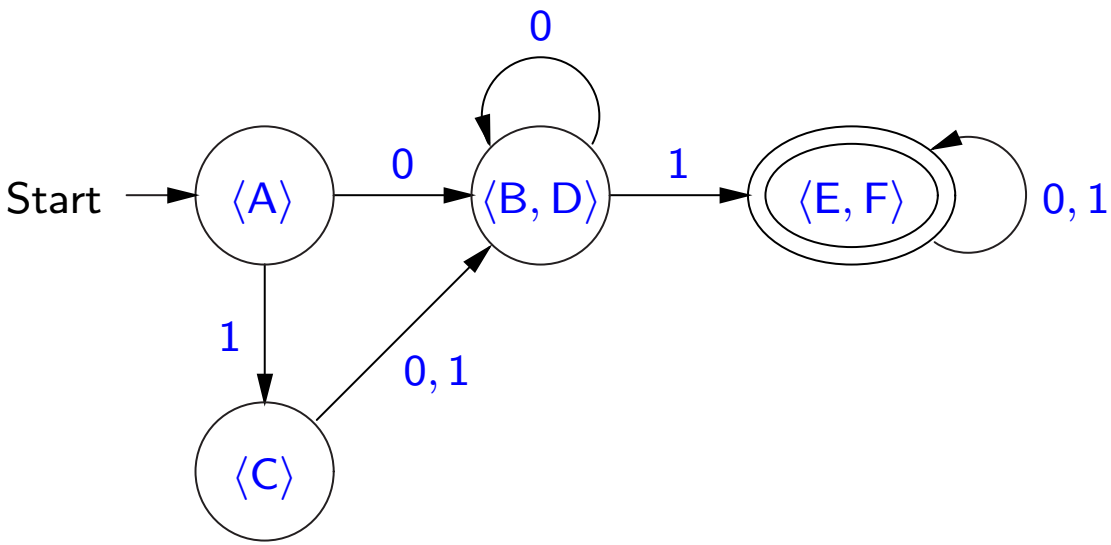
- Since $\{\mathrm{A}\} \in Z$ and $[\delta_{M'}(\mathrm{A}, 0)] = [\mathrm{B}] = \{\mathrm{B}, \mathrm{D}\}$, we have that $(\langle \mathrm{A} \rangle, 0, \langle \mathrm{B}, \mathrm{D} \rangle) \in T_N$.
  Since $\{\mathrm{A}\} \in Z$ and $[\delta_{M'}(\mathrm{A}, 1)] = [\mathrm{C}] = \{\mathrm{C}\}$, we have that $(\langle \mathrm{A} \rangle, 1, \langle \mathrm{C} \rangle) \in T_N$.

Example Minimization

- Since $\{C\} \in Z$ and $[\delta_{M'}(C, 0)] = [D] = \{B, D\}$, we have that $(\langle C \rangle, 0, \langle B, D \rangle) \in T_N$.
  Since $\{C\} \in Z$ and $[\delta_{M'}(C, 1)] = [D] = \{B, D\}$, we have that $(\langle C \rangle, 1, \langle B, D \rangle) \in T_N$.
- Since $\{B, D\} \in Z$ and $[\delta_{M'}(B, 0)] = [D] = \{B, D\}$, we have that $(\langle B, D \rangle, 0, \langle B, D \rangle) \in T_N$.
  Since $\{B, D\} \in Z$ and $[\delta_{M'}(B, 1)] = [E] = \{E, F\}$, we have that $(\langle B, D \rangle, 1, \langle E, F \rangle) \in T_N$.
- Since $\{E, F\} \in Z$ and $[\delta_{M'}(E, 0)] = [F] = \{E, F\}$, we have that $(\langle E, F \rangle, 0, \langle E, F \rangle) \in T_N$.
  Since $\{E, F\} \in Z$ and $[\delta_{M'}(E, 1)] = [F] = \{E, F\}$, we have that $(\langle E, F \rangle, 1, \langle E, F \rangle) \in T_N$.

Example Minimization

Thus our DFA N is:



Correctness of Minimization

Back in the general case, we have the following lemmas.

Lemma 3.13.7

(1) For all $q \in Q_{M'}$, $\overline{[q]} \in A_N$ iff $q \in A_{M'}$.

(2) For all $q \in Q_{M'}$ and $a \in$ **alphabet** M', $\delta_N(\overline{[q]}, a) = \overline{[\delta_{M'}(q, a)]}$.

(3) For all $q \in Q_{M'}$ and $w \in$ (**alphabet** M')*, $\delta_N(\overline{[q]}, w) = \overline{[\delta_{M'}(q, w)]}$.

(4) For all $w \in$ (**alphabet** M')*, $\delta_N(s_N, w) = \overline{[\delta_{M'}(s_{M'}, w)]}$.

Proof. (1) and (2) follow easily by Lemma 3.13.5(2)–(3). Part (3) follows from Part (2) by left string induction. For Part (4), suppose $w \in$ (**alphabet** M')*. By Part (3), we have

$$\delta_N(s_N, w) = \delta_N(\overline{[s_{M'}]}, w) = \overline{[\delta_{M'}(s_{M'}, w)]}.$$

□

Lemma 3.13.8
$L(N) = L(M')$.

Proof. Suppose $w \in L(N)$. Then $w \in (\textbf{alphabet}\, N)^* = (\textbf{alphabet}\, M')^*$ and $\delta_N(s_N, w) \in A_N$. By Lemma 3.13.7(4), we have that

$$\overline{[\delta_{M'}(s_{M'}, w)]} = \delta_N(s_N, w) \in A_N,$$

so that $\delta_{M'}(s_{M'}, w) \in A_{M'}$, by Lemma 3.13.7(1). Thus $w \in L(M')$.
Suppose $w \in L(M')$. Then $w \in (\textbf{alphabet}\, M')^* = (\textbf{alphabet}\, N)^*$ and $\delta_{M'}(s_{M'}, w) \in A_{M'}$. By Lemma 3.13.7(1) and (4), we have that

$$\delta_N(s_N, w) = \overline{[\delta_{M'}(s_{M'}, w)]} \in A_N.$$

Hence $w \in L(N)$. □

Correctness of Minimization

Lemma 3.13.9

N is deterministically simplified.

Proof. To see that all elements of N are reachable, suppose $q \in Q_{M'}$. Because M' is deterministically simplified, there is a $w \in (\textbf{alphabet}\ M')^*$ such that $q = \delta_{M'}(s_{M'}, w)$. Thus $\delta_N(s_N, w) = \overline{[\delta_{M'}(s_{M'}, w)]} = \overline{[q]}$.

Next, we show that, for all $q \in Q_{M'}$, if q is live, then $\overline{[q]}$ is live. Suppose $q \in Q_{M'}$ is live, so there is a $w \in (\textbf{alphabet}\ M')^*$ such that $\delta_{M'}(q, w) \in A_{M'}$. Thus $\delta_N(\overline{[q]}, w) = \overline{[\delta_{M'}(q, w)]} \in A_N$, showing that $\overline{[q]}$ is live.

Thus, we have that, for all $q \in Q_{M'}$, if $\overline{[q]}$ is dead, then q is dead. But, M' has at most one dead state, and thus we have that N has at most one dead state. $\square$

Correctness of Minimization

Lemma 3.13.10
Suppose N' *is a DFA such that* $N' \approx M'$, **alphabet** $N' =$ **alphabet** M' *and* $|Q_{N'}| \leq |Q_N|$. *Then* N' *is isomorphic to* N.

Proof. We have that $L(N') = L(M')$. And the states of M' and N are all reachable.

Let the relation h between $Q_{N'}$ and Q_N be

$$\{ (\delta_{N'}(s_{N'}, w), \delta_N(s_N, w)) \mid w \in (\textbf{alphabet}\, M')^* \}.$$

Since every state of N is reachable, it follows that **range** $h = Q_N$.

Correctness of Minimization

Proof (cont.). To see that h is a function, suppose $x, y \in (\textbf{alphabet}\ M')^*$ and $\delta_{N'}(s_{N'}, x) = \delta_{N'}(s_{N'}, y)$. We must show that $\delta_N(s_N, x) = \delta_N(s_N, y)$. Since $\delta_N(s_N, x) = [\delta_{M'}(s_{M'}, x)]$ and $\delta_N(s_N, y) = [\delta_{M'}(s_{M'}, y)]$, it will suffice to show that $(\delta_{M'}(s_{M'}, x), \delta_{M'}(s_{M'}, y)) \in Y$. By Lemma 3.13.5(1), it will suffice to show that, $\delta_{M'}(\delta_{M'}(s_{M'}, x), z) \in A_{M'}$ iff $\delta_{M'}(\delta_{M'}(s_{M'}, y), z) \in A_{M'}$, for all $z \in (\textbf{alphabet}\ M')^*$. Suppose $z \in (\textbf{alphabet}\ M')^*$. We must show that $\delta_{M'}(\delta_{M'}(s_{M'}, x), z) \in A_{M'}$ iff $\delta_{M'}(\delta_{M'}(s_{M'}, y), z) \in A_{M'}$

Correctness of Minimization

Proof (cont.). We will show the "only if" direction, the other direction being similar. Suppose $\delta_{M'}(\delta_{M'}(s_{M'}, x), z) \in A_{M'}$. We must show that $\delta_{M'}(\delta_{M'}(s_{M'}, y), z) \in A_{M'}$. Because $\delta_{M'}(s_{M'}, xz) = \delta_{M'}(\delta_{M'}(s_{M'}, x), z) \in A_{M'}$, we have that $xz \in L(M') = L(N')$. Since $xz \in L(N')$ and $\delta_{N'}(s_{N'}, x) = \delta_{N'}(s_{N'}, y)$, we have that

$$\begin{aligned}\delta_{N'}(s_{N'}, yz) &= \delta_{N'}(\delta_{N'}(s_{N'}, y), z) = \delta_{N'}(\delta_{N'}(s_{N'}, x), z) \\ &= \delta_{N'}(s_{N'}, xz) \in A_{N'},\end{aligned}$$

so that $yz \in L(N') = L(M')$. Hence $\delta_{M'}(\delta_{M'}(s_{M'}, y), z) = \delta_{M'}(s_{M'}, yz) \in A_{M'}$.

Correctness of Minimization

Proof (cont.). Because h is a function and $\textbf{range}\, h = Q_N$, we have that $|Q_N| \leq |\textbf{domain}\, h| \leq Q_{N'}$. But $|Q_{N'}| \leq |Q_N|$, and thus $|Q_{N'}| = |Q_N|$. Because $Q_{N'}$ and Q_N are finite, it follows that $\textbf{domain}\, h = Q_{N'}$ and h is injective, so that h is a bijection from $Q_{N'}$ to Q_N. Thus, every state of N' is reachable, and, for all $w \in (\textbf{alphabet}\, M')^* = (\textbf{alphabet}\, N)^* = (\textbf{alphabet}\, N')^*$, $h(\delta_{N'}(s_{N'}, w)) = \delta_N(s_N, w)$. The remainder of the proof that h is an isomorphism from N' to N is easy. □

Minimization Function and Specification

We define a function $\textbf{minimize} \in \textbf{DFA} \to \textbf{DFA}$ by: $\textbf{minimize}\, M$ is the result of running the above algorithm on input M.

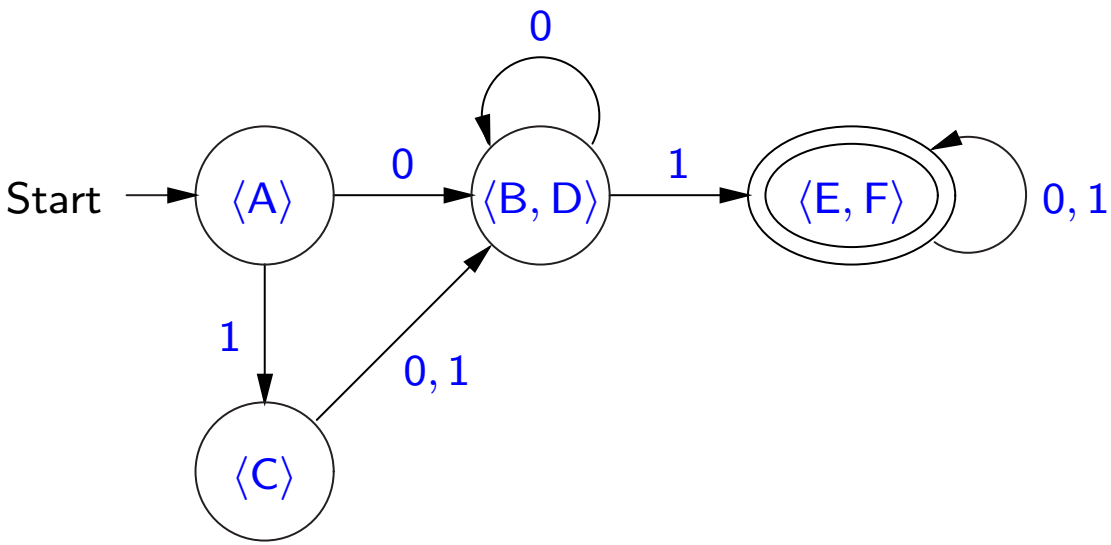
Putting the above results together, we have the following theorem:

Theorem 3.13.12
For all $M \in \textbf{DFA}$*:*

- $\textbf{minimize}\, M \approx M$;
- $\textbf{alphabet}(\textbf{minimize}\, M) = \textbf{alphabet}(L(M))$;
- $\textbf{minimize}\, M$ *is deterministically simplified; and*
- *for all* $N \in \textbf{DFA}$*, if* $N \approx M$, $\textbf{alphabet}\, N = \textbf{alphabet}(L(M))$ *and* $|Q_N| \leq |Q_{\textbf{minimize}\, M}|$*, then* N *is isomorphic to* $\textbf{minimize}\, M$.

Minimization Specification

Thus



is, up to isomorphism, the only four-or-fewer state DFA with alphabet $\{0, 1\}$ that is equivalent to M.

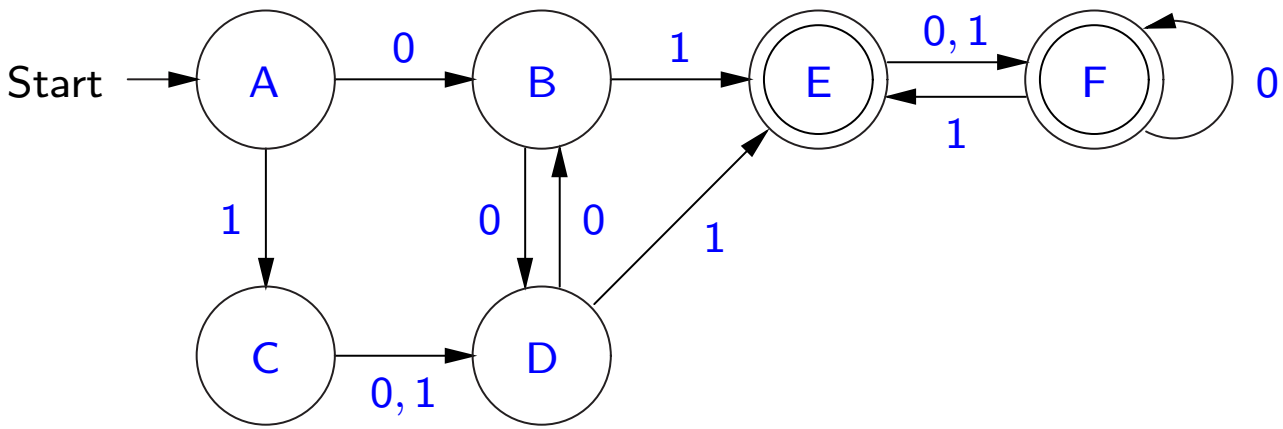
Minimization in Forlan

The Forlan module DFA includes the function

```
val minimize : dfa -> dfa
```

for minimizing DFAs.

For example, if `dfa` of type `dfa` is bound to our example DFA



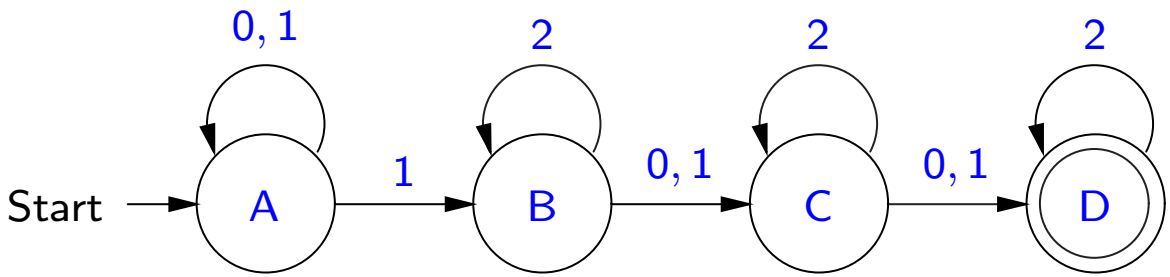
then we can minimize the alphabet size and number of states of `dfa` as follows.

Minimization in Forlan

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

Revisiting NFA-to-DFA Conversion Example

Finally, let's revisit an example from Section 3.11. Suppose `nfa` is the 4-state NFA



As we saw, our NFA-to-DFA conversion algorithm converts `nfa` to a DFA `dfa` with 16 states:

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

We can now use Forlan to verify that there is no DFA with fewer than 16 states that accepts the same language as `nfa`:

Revisiting NFA-to-DFA Conversion Example

```
- val dfa' = DFA.minimize dfa;
val dfa' = - : dfa
- DFA.isomorphic(dfa', dfa);
val it = true : bool
- DFA.numStates dfa';
val it = 16 : int
```

Thus we have an example where the smallest DFA accepting a language requires exponentially more states than the smallest NFA accepting that language. (This is true even though we haven't proven that an NFA must have at least 4 states to accept the same language as `nfa`.)