

Section 3.12: Equivalence-testing and Minimization of Deterministic Finite Automata

In this section, we give algorithms for:

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

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

Copyright © 2003–6 Alley Stoughton

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

The L^AT_EX source of these slides, the associated book, and the distribution of the Forlan toolset are available on the WWW at <http://people.cis.ksu.edu/~stough/forlan/>.

1

(3.12) Testing the Equivalence of DFAs

Suppose M and N are DFAs. To check whether they are equivalent, we can proceed as follows.

First, we need to convert M and N into DFAs with identical alphabets. Let $\Sigma = \text{alphabet}(M) \cup \text{alphabet}(N)$, and define the DFAs M' and N' by:

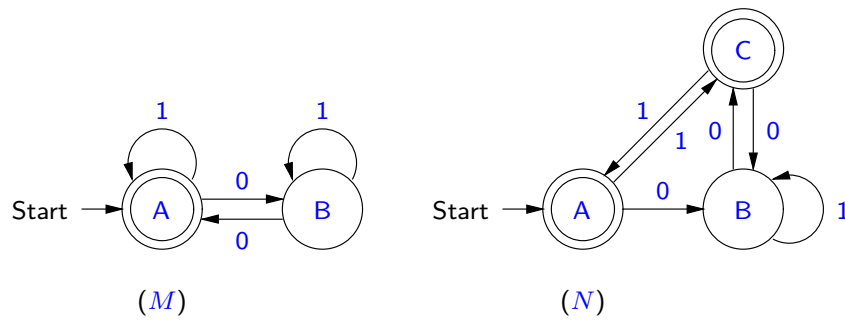
$$\begin{aligned}M' &= \text{determSimplify}(M, \Sigma), \\N' &= \text{determSimplify}(N, \Sigma).\end{aligned}$$

Since $\text{alphabet}(L(M)) \subseteq \text{alphabet}(M) \subseteq \Sigma$, we have that $\text{alphabet}(M') = \text{alphabet}(L(M)) \cup \Sigma = \Sigma$. Similarly, $\text{alphabet}(N') = \Sigma$. Furthermore, $M' \approx M$ and $N' \approx N$, so that it will suffice to determine whether M' and N' are equivalent.

2

(3.12) Equivalence-testing (Cont.)

For example, if M and N are the DFAs



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

3

(3.12) Equivalence-testing (Cont.)

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

- $(s_{M'}, s_{N'}) \in X$;
- 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

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

4

(3.12) Equivalence-testing (Cont.)

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

Lemma 3.12.1

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

Proof. By left string induction on w . \square

Lemma 3.12.2

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 . \square

Finally, we check that, for all $(q, r) \in X$,

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

If this is true, we say that the machines are equivalent; otherwise we say they are not equivalent.

5

(3.12) Equivalence-testing (Cont.)

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 3.12.1 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 3.12.2, 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 $L(M') \neq L(N')$.

6

(3.12) Equivalence-testing (Cont.)

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

We can easily modify our algorithm so that, when two machines are not equivalent, it explains why:

- 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.

7

(3.12) 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
```

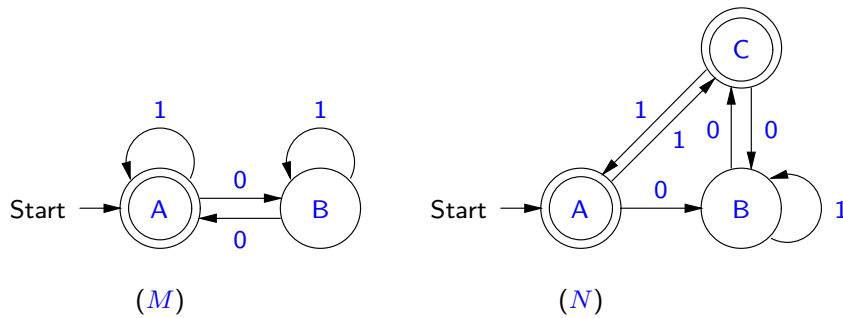
The function `relationship` figures out the relationship between the languages accepted by two DFAs (are they equal, is one a proper subset of the other, is neither a subset of the other), and supplies minimum-length counterexamples to justify negative answers. The function `subset` tests whether its 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 `Reg.simplify` (see Section 3.2).

8

(3.12) Forlan Examples

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



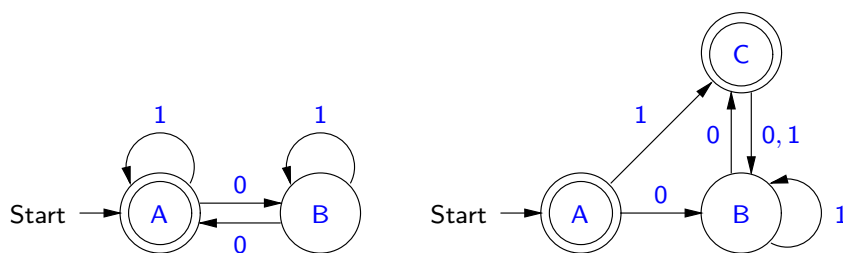
We can verify that these machines are equivalent as follows:

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

9

(3.12) Forlan Examples (Cont.)

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
```

(3.12) Forlan Examples (Cont.)

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
```

11

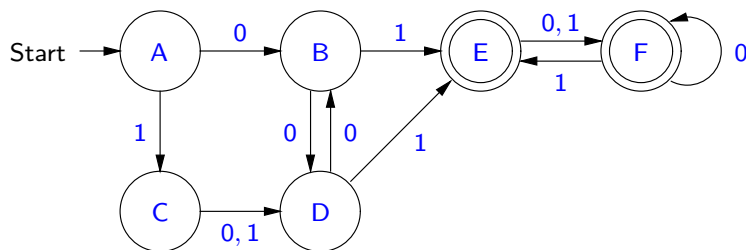
(3.12) Minimization of DFAs

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

First, we minimize the size of M 's alphabet, and make the automaton be deterministically simplified, by letting

$M' = \mathbf{determSimplify}(M, \emptyset)$. Thus $M' \approx M$ and $\mathbf{alphabet}(M') = \mathbf{alphabet}(L(M))$.

For example, if M is the DFA



then $M' = M$.

12

(3.12) Unmergable States

Next, we let X be the least subset of $Q_{M'} \times Q_{M'}$ such that:

- (1) $A_{M'} \times (Q_{M'} - A_{M'}) \subseteq X$;
- (2) $(Q_{M'} - A_{M'}) \times A_{M'} \subseteq X$;
- (3) for all $q, q', r, r' \in Q_{M'}$ and $a \in \mathbf{alphabet}(M')$, 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”. The idea of (1) and (2) is that an accepting state can never be merged with a non-accepting state. And (3) says that if q and r can't be merged, and we can get from q' to q by processing an a , and from r' to r by processing an a , then q' and r' also can't be merged—since if we merged q' and r' , there would have to be an a -transition from the merged state to the merging of q and r .

13

(3.12) Unmergable States (Cont.)

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) .

14

(3.12) Unmergable States (Cont.)

- 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.

15

(3.12) Unmergable States (Cont.)

- 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.

16

(3.12) Unmergable States (Cont.)

- 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 .

17

(3.12) Unmergable States (Cont.)

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.

18

(3.12) Unmergable States (Cont.)

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) .

19

(3.12) Mergable States

Going back to the general case, we now let the relation $Y = (Q_{M'} \times Q_{M'}) - X$. It turns out that Y is reflexive on $Q_{M'}$, symmetric and transitive, i.e., it is what is called an equivalence relation on $Q_{M'}$. We read " $(q, r) \in Y$ " as " q and r can be merged".

Back with our example, we have that Y is

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

20

(3.12) 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 3.10, we write \overline{P} for the result of coding a finite set of symbols P as a symbol. E.g., $\overline{\{B, A\}} = \langle A, B \rangle$.

If $q \in Q_{M'}$, we write $[q]$ for $\{r \in Q_{M'} \mid (r, q) \in Y\}$, which is called the *equivalence class* of q .

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.

21

(3.12) Definition of Minimized DFA

Let $Z = \{[q] \mid q \in Q_{M'}\}$.

In the case of our example, Z is

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

We define our DFA N as follows:

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

(In the definitions of A_N and T_N any element of P could be substituted for $\min(P)$.)

22

(3.12) Example Minimization (Cont.)

In the case of our example, we have that

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

We compute the elements of T_N as follows.

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

23

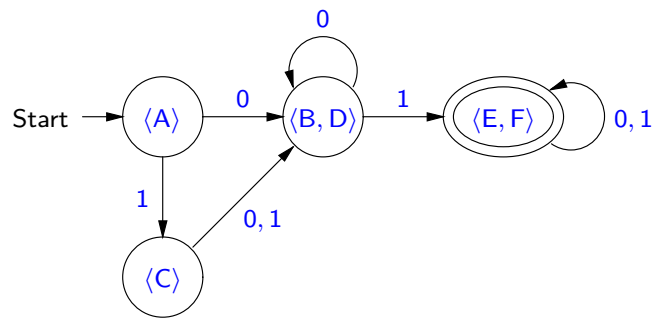
(3.12) Example Minimization (Cont.)

- 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 $(\langle E, F \rangle, 1, \langle E, F \rangle) \in T_N$.

24

(3.12) Example Minimization (Cont.)

Thus our DFA N is:



25

(3.12) Minimization Function and Specification

We define a function $\text{minimize} \in \text{DFA} \rightarrow \text{DFA}$ by: $\text{minimize}(M)$ is the result of running the above algorithm on input M .

We have the following theorem:

Theorem 3.12.11

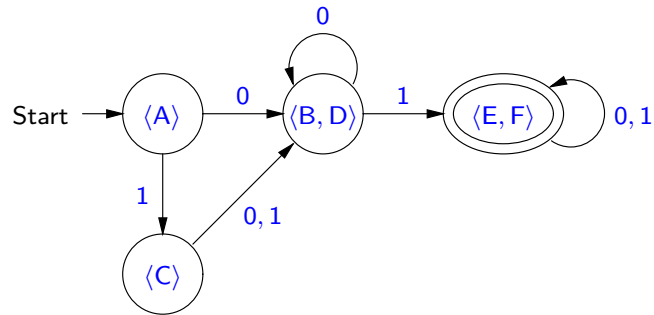
For all $M \in \text{DFA}$:

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

26

(3.12) Minimization Specification (Cont.)

Thus



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

27

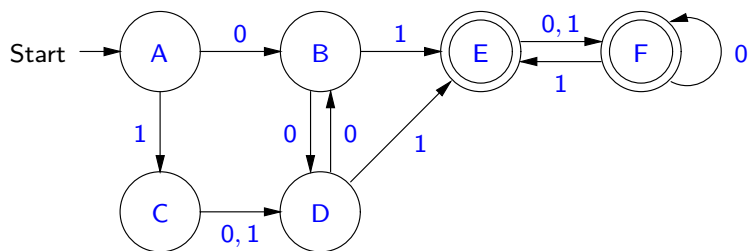
(3.12) Minimization in Forlan

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.

28

(3.12) Minimization in Forlan (Cont.)

```
- 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
```

29

(3.12) An Alternative Approach to Synthesizing DFAs

Because of DFA minimization plus the operations on automata and regular expressions of Section 3.11, we now have an alternative way of synthesizing DFAs.

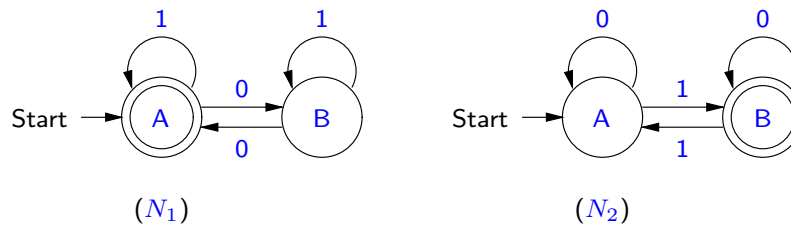
For example, suppose we wish to find a DFA M such that $L(M) = X$, where $X = \{w \in \{0,1\}^* \mid w \text{ has an even number of 0's or an odd number of 1's}\}$.

First, we can note that $X = Y_1 \cup Y_2$, where $Y_1 = \{w \in \{0,1\}^* \mid w \text{ has an even number of 0's}\}$ and $Y_2 = \{w \in \{0,1\}^* \mid w \text{ has an odd number of 1's}\}$. Since we have a union operation on EFAs (Forlan doesn't provide a union operation on DFAs), if we can find EFAs accepting Y_1 and Y_2 , we can combine them into a EFA that accepts X . Then we can convert this EFA to a DFA, and then minimize the DFA.

30

(3.12) Synthesizing DFAs (Cont.)

Let N_1 and N_2 be the DFAs



It is easy to prove that $L(N_1) = Y_1$ and $L(N_2) = Y_2$. Let M be the DFA

renameStatesCanonically(minimize(N)),

where N is the DFA

nfaToDFA(efaToNFA(union(N_1, N_2))).

31

(3.12) Synthesizing DFAs (Cont.)

Then

$$\begin{aligned}
 L(M) &= L(\text{renameStatesCanonically}(\text{minimize}(N))) \\
 &= L(\text{minimize}(N)) \\
 &= L(N) \\
 &= L(\text{nfaToDFA}(\text{efaToNFA}(\text{union}(N_1, N_2)))) \\
 &= L(\text{efaToNFA}(\text{union}(N_1, N_2))) \\
 &= L(\text{union}(N_1, N_2)) \\
 &= L(N_1) \cup L(N_2) \\
 &= Y_1 \cup Y_2 \\
 &= X,
 \end{aligned}$$

showing that M is correct.

32

(3.12) Synthesizing DFAs (Cont.)

But how do we figure out what the components of M are, so that, e.g., we can draw M ? In a simple case like this, we could apply the definitions `union`, `efaToNFA`, `nfaToDFA`, `minimize` and `renameStatesCanonically`, and work out the answer. But, for more complex examples, there would be far too much detail involved for this to be a practical approach.

Instead, we can use Forlan to compute the answer. Suppose `dfa1` and `dfa2` of type `dfa` are N_1 and N_2 , respectively. Then we can proceed as follows.

33

(3.12) Synthesizing DFAs (Cont.)

```
- val efa = EFA.union(injDFAToEFA dfa1, injDFAToEFA dfa2);
val efa = - : efa
- val dfa' = nfaToDFA(efaToNFA efa);
val dfa' = - : dfa
- DFA.numStates dfa';
val it = 5 : int
- val dfa =
=       DFA.renameStatesCanonically(DFA.minimize dfa');
val dfa = - : dfa
- DFA.numStates dfa;
val it = 4 : int
```

34

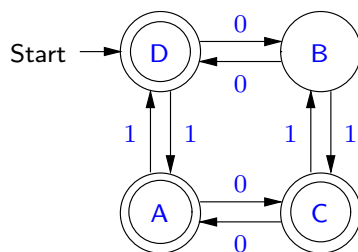
(3.12) Synthesizing DFAs (Cont.)

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

35

(3.12) Synthesizing DFAs (Cont.)

Thus M is:



Of course, this claim assumes that Forlan is correctly implemented.

36