

## Section 3.9: Nondeterministic Finite Automata

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

---

Copyright © 2003–4 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  $\text{\LaTeX}$  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

### (3.9) Introduction (Cont.)

A *nondeterministic finite automaton* (NFA)  $M$  is a finite automaton such that

$$T_M \subseteq \{ (q, x, r) \mid q, r \in \mathbf{Sym} \text{ and } x \in \mathbf{Str} \text{ and } |x| = 1 \}.$$

In other words, an FA is an NFA iff every string of every transition of the FA has a single symbol.

For example,  $(A, 1, B)$  is a legal NFA transition, but  $(A, \%, B)$  and  $(A, 11, B)$  are not legal.

We write  $\mathbf{NFA}$  for the set of all nondeterministic finite automata. Thus  $\mathbf{NFA} \subsetneq \mathbf{EFA} \subsetneq \mathbf{FA}$ .

2

## (3.9) Properties of NFAs

### Proposition 3.9.1

Suppose  $M$  is an NFA. For all  $p, q \in Q_M$ , if  $q \in \Delta(\{p\}, \%)$ , then  $q = p$ .

### Proposition 3.9.2

Suppose  $M$  is an NFA. For all  $p, r \in Q_M$ ,  $a \in \mathbf{Sym}$  and  $x \in \mathbf{Str}$ , if  $r \in \Delta_M(\{p\}, ax)$ , then there is a  $q \in Q_M$  such that  $(p, a, q) \in T_M$  and  $r \in \Delta_M(\{q\}, x)$ .

### Proposition 3.9.3

Suppose  $M$  is an NFA. For all  $p, r \in Q_M$ ,  $a \in \mathbf{Sym}$  and  $x \in \mathbf{Str}$ , if  $r \in \Delta_M(\{p\}, xa)$ , then there is a  $q \in Q_M$  such that  $q \in \Delta_M(\{p\}, x)$  and  $(q, a, r) \in T_M$ .

None of these propositions hold for arbitrary EFAs.

3

## (3.9) Properties of NFAs (Cont.)

The following proposition obviously holds.

### Proposition 3.9.4

Suppose  $M$  is an NFA.

- For all  $N \in \mathbf{FA}$ , if  $M \text{ iso } N$ , then  $N$  is an NFA.
- For all bijections  $f$  from  $Q_M$  to some set of symbols,  $\text{renameStates}(M, f)$  is an NFA.
- $\text{renameStatesCanonically}(M)$  is an NFA.
- $\text{simplify}(M)$  is an NFA.

4

## (3.9) Correctness Proofs for NFAs

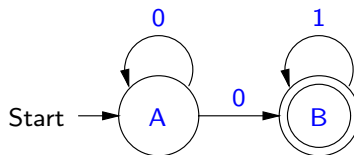
Since none of the strings of the transitions of an NFA are  $\epsilon$ , when proving  $L(M) \subseteq X$ , for an NFA  $M$  and a language  $X$ , we can always use strong string induction, instead of having to resort to using induction on the length of labeled paths.

In fact, since every string of every transition consists of a single symbol, we can use left string induction rather than strong string induction.

5

## (3.9) Example Correctness Proof

Let  $M$  be the NFA



To show that  $L(M) = \{0\}^*\{0\}\{1\}^*$ , it will suffice to show that  $L(M) \subseteq \{0\}^*\{0\}\{1\}^*$  and  $\{0\}^*\{0\}\{1\}^* \subseteq L(M)$ . We will show the proof of  $L(M) \subseteq \{0\}^*\{0\}\{1\}^*$ .

Since  $\text{alphabet}(M) = \{0, 1\}$ , it will suffice to show that, for all  $w \in \{0, 1\}^*$ ,

- (A) if  $A \in \Delta(\{A\}, w)$ , then  $w \in \{0\}^*$ ;
- (B) if  $B \in \Delta(\{A\}, w)$ , then  $w \in \{0\}^*\{0\}\{1\}^*$ .

We proceed by left string induction.

6

### (3.9) Example Correctness Proof (Cont.)

(Basis Step) We must show that:

(A) if  $A \in \Delta(\{A\}, \%)$ , then  $\% \in \{0\}^*$ ;

(B) if  $B \in \Delta(\{A\}, \%)$ , then  $\% \in \{0\}^* \{0\} \{1\}^*$ .

(A) Suppose  $A \in \Delta(\{A\}, \%)$ . Then  $\% \in \{0\}^*$ .

(B) Suppose  $B \in \Delta(\{A\}, \%)$ . By Proposition 3.9.1, we have that  $B = A$ —contradiction. Thus  $\% \in \{0\}^* \{0\} \{1\}^*$ .

7

### (3.9) Example Correctness Proof (Cont.)

(Inductive Step) Suppose  $a \in \{0, 1\}$  and  $w \in \{0, 1\}^*$ . Assume the inductive hypothesis:

(A) if  $A \in \Delta(\{A\}, w)$ , then  $w \in \{0\}^*$ ;

(B) if  $B \in \Delta(\{A\}, w)$ , then  $w \in \{0\}^* \{0\} \{1\}^*$ .

We must show that:

(A) if  $A \in \Delta(\{A\}, wa)$ , then  $wa \in \{0\}^*$ ;

(B) if  $B \in \Delta(\{A\}, wa)$ , then  $wa \in \{0\}^* \{0\} \{1\}^*$ .

8

### (3.9) Example Correctness Proof (Cont.)

(A) Suppose  $A \in \Delta(\{A\}, wa)$ . We must show that  $wa \in \{0\}^*$ . By Proposition 3.9.3, there is a  $q \in Q$  such that  $q \in \Delta(\{A\}, w)$  and  $(q, a, A) \in T$ . Thus  $q = A$  and  $a = 0$ , so that  $A \in \Delta(\{A\}, w)$ . By part (A) of the inductive hypothesis, we have that  $w \in \{0\}^*$ . Thus  $wa = w0 \in \{0\}^*\{0\} \subseteq \{0\}^*$ .

9

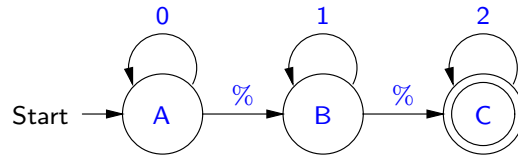
### (3.9) Example Correctness Proof (Cont.)

(B) Suppose  $B \in \Delta(\{A\}, wa)$ . We must show that  $wa \in \{0\}^*\{0\}\{1\}^*$ . By Proposition 3.9.3, there is a  $q \in Q$  such that  $q \in \Delta(\{A\}, w)$  and  $(q, a, B) \in T$ . There are two subcases to consider.

- Suppose  $q = A$  and  $a = 0$ . Then  $A \in \Delta(\{A\}, w)$ . Part (A) of the inductive hypothesis tell us that  $w \in \{0\}^*$ . Thus  $wa = w0 \in \{0\}^*\{0\}\{1\}^*$ .
- Suppose  $q = B$  and  $a = 1$ . Then  $B \in \Delta(\{A\}, w)$ . Part (B) of the inductive hypothesis tell us that  $w \in \{0\}^*\{0\}\{1\}^*$ . Thus  $wa = w1 \in \{0\}^*\{0\}\{1\}^*\{1\} \subseteq \{0\}^*\{0\}\{1\}^*$ .

## (3.9) Converting EFAs to NFAs

Suppose  $M$  is the EFA



To convert  $M$  into an equivalent NFA, we will have to:

- replace the transitions  $(A, \%, B)$  and  $(B, \%, C)$  with legal transitions (for example, because of the valid labeled path

$$A \xrightarrow{\%} B \xrightarrow{1} B \xrightarrow{\%} C,$$

we will add the transition  $(A, 1, C)$ ;

- make (at least)  $A$  be an accepting state (so that  $\%$  is accepted by the NFA).

11

## (3.9) The Empty-closure of a Set of States

Before defining a general procedure for converting EFAs to NFAs, we first say what we mean by the empty-closure of a set of states.

Suppose  $M$  is a finite automaton and  $P \subseteq Q_M$ . The *empty-closure* of  $P$  ( $\mathbf{emptyClose}_M(P)$ ) is the least subset  $X$  of  $Q_M$  such that

- $P \subseteq X$ ;
- for all  $q, r \in Q_M$ , if  $q \in X$  and  $(q, \%, r) \in T_M$ , then  $r \in X$ .

We sometimes abbreviate  $\mathbf{emptyClose}_M(P)$  to  $\mathbf{emptyClose}(P)$ .

12

## (3.9) Empty-closure (Cont.)

For example, if  $M$  is our example EFA and  $P = \{A\}$ , then:

- $A \in X$ ;
- $B \in X$ , since  $A \in X$  and  $(A, \%, B) \in T_M$ ;
- $C \in X$ , since  $B \in X$  and  $(B, \%, C) \in T_M$ .

Thus  $\text{emptyClose}(P) = \{A, B, C\}$ .

### Proposition 3.9.5

Suppose  $M$  is a finite automaton. For all  $P \subseteq Q_M$ ,  
 $\text{emptyClose}_M(P) = \Delta_M(P, \%)$ .

In other words,  $\text{emptyClose}_M(P)$  is all of the states that can be reached from elements of  $P$  by sequences of empty moves.

13

## (3.9) Backwards Empty-closure

Suppose  $M$  is a finite automaton and  $P \subseteq Q_M$ . The *backwards empty-closure* of  $P$  ( $\text{emptyCloseBackwards}_M(P)$ ) is the least subset  $X$  of  $Q_M$  such that

- $P \subseteq X$ ;
- for all  $q, r \in Q_M$ , if  $r \in X$  and  $(q, \%, r) \in T_M$ , then  $q \in X$ .

14

## (3.9) Backwards Empty-closure (Cont.)

For example, if  $M$  is our example EFA and  $P = \{C\}$ , then:

- $C \in X$ ;
- $B \in X$ , since  $C \in X$  and  $(B, \%, C) \in T_M$ ;
- $A \in X$ , since  $B \in X$  and  $(A, \%, B) \in T_M$ .

Thus  $\text{emptyCloseBackwards}(P) = \{A, B, C\}$ .

### Proposition 3.9.6

Suppose  $M$  is a finite automaton. For all  $P \subseteq Q_M$ ,

$$\text{emptyCloseBackwards}_M(P) = \{q \in Q_M \mid \Delta_M(\{q\}, \%) \cap P \neq \emptyset\}.$$

In other words,  $\text{emptyCloseBackwards}_M(P)$  is all of the states from which it is possible to reach elements of  $P$  by sequences of empty moves.

15

## (3.9) Conversion Algorithm

We define a function  $\text{efaToNFA} \in \mathbf{EFA} \rightarrow \mathbf{NFA}$  that converts EFAs into NFAs by saying that  $\text{efaToNFA}(M)$  is the NFA  $N$  such that:

- $Q_N = Q_M$ ;
- $s_N = s_M$ ;
- $A_N = \text{emptyCloseBackwards}(A_M)$ ;
- $T_N$  is the set of all triples  $(q', a, r')$  such that  $q', r' \in Q_M$ ,  $a \in \mathbf{Sym}$ , and there are  $q, r \in Q_M$  such that:
  - $(q, a, r) \in T_M$ ;
  - $q' \in \text{emptyCloseBackwards}(\{q\})$ ; and
  - $r' \in \text{emptyClose}(\{r\})$ .

16

### (3.9) Conversion Algorithm (Cont.)

To compute the set  $T_N$ , we process each transition  $(q, x, r)$  of  $M$  as follows. If  $x = \%$ , then we generate no transitions. Otherwise, our transition is  $(q, a, r)$  for some symbol  $a$ . We then compute the backwards empty-closure of  $\{q\}$ , and call the result  $X$ , and compute the (forwards) empty-closure of  $\{r\}$ , and call the result  $Y$ . We then add all of the elements of

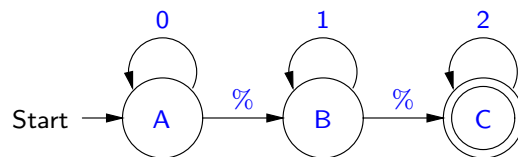
$$\{(q', a, r') \mid q' \in X \text{ and } r' \in Y\}$$

to  $T_N$ .

17

### (3.9) Conversion Example

Let  $M$  be our example EFA



and let  $N = \text{efaToNFA}(M)$ . Then

- $Q_N = Q_M = \{A, B, C\}$ ;
- $s_N = s_M = A$ ;
- $A_N = \text{emptyCloseBackwards}(A_M) = \text{emptyCloseBackwards}(\{C\}) = \{A, B, C\}$ .

18

### (3.9) Conversion Example (Cont.)

Now, let's work out what  $T_N$  is, by processing each of  $M$ 's transitions.

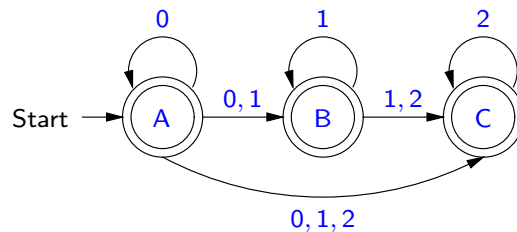
- From the transitions  $(A, \%, B)$  and  $(B, \%, C)$ , we get no elements of  $T_N$ .
- Consider the transition  $(A, 0, A)$ . Since  $\text{emptyCloseBackwards}(\{A\}) = \{A\}$  and  $\text{emptyClose}(\{A\}) = \{A, B, C\}$ , we add  $(A, 0, A)$ ,  $(A, 0, B)$  and  $(A, 0, C)$  to  $T_N$ .
- Consider the transition  $(B, 1, B)$ . Since  $\text{emptyCloseBackwards}(\{B\}) = \{A, B\}$  and  $\text{emptyClose}(\{B\}) = \{B, C\}$ , we add  $(A, 1, B)$ ,  $(A, 1, C)$ ,  $(B, 1, B)$  and  $(B, 1, C)$  to  $T_N$ .

19

### (3.9) Conversion Example (Cont.)

- Consider the transition  $(C, 2, C)$ . Since  $\text{emptyCloseBackwards}(\{C\}) = \{A, B, C\}$  and  $\text{emptyClose}(\{C\}) = \{C\}$ , we add  $(A, 2, C)$ ,  $(B, 2, C)$  and  $(C, 2, C)$  to  $T_N$ .

Thus our NFA  $N$  is



20

## (3.9) Specification of Conversion Function

### Theorem 3.9.7

For all  $M \in \mathbf{EFA}$ :

- $\mathbf{efaToNFA}(M) \approx M$ ; and
- $\mathbf{alphabet}(\mathbf{efaToNFA}(M)) = \mathbf{alphabet}(M)$ .

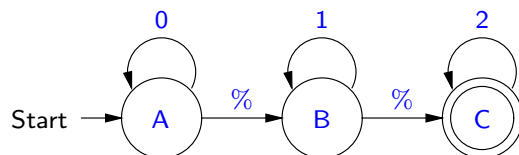
## (3.9) Computing Empty-closures in Forlan

The Forlan module **FA** defines the following functions for computing forwards and backwards empty-closures:

```
val emptyClose          : fa -> sym set -> sym set
val emptyCloseBackwards : fa -> sym set -> sym set
```

## (3.9) Empty-closures in Forlan (Cont.)

For example, if `fa` is bound to the finite automaton



then we can compute the empty-closure of `{A}` as follows:

```
- SymSet.output("", FA.emptyClose fa (SymSet.input ""));
@ A
@ .
A, B, C
val it = () : unit
```

23

## (3.9) Processing NFAs in Forlan

The Forlan module `NFA` defines an abstract type `nfa` (in the top-level environment) of nondeterministic finite automata, along with various functions for processing NFAs.

Values of type `nfa` are implemented as values of type `fa`, and the module `NFA` provides the following injection and projection functions:

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

The functions `injToFA`, `injToEFA`, `projFromFA` and `projFromEFA` are available in the top-level environment as `injNFAToFA`, `injNFAToEFA`, `projFAToNFA` and `projEFAToNFA`, respectively.

24

## (3.9) Processing NFAs in Forlan (Cont.)

The module `NFA` also defines the functions:

```
val input    : string -> nfa
val fromEFA : efa  -> nfa
```

The function `input` is used to input an NFA, and the function `fromEFA` corresponds to our conversion function `efaToNFA`, and is available in the top-level environment with that name:

```
val efaToNFA : efa -> nfa
```

25

## (3.9) Processing NFAs in Forlan (Cont.)

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

```
val output           : string * nfa -> unit
val numStates       : nfa -> int
val numTransitions  : nfa -> int
val alphabet        : nfa -> sym set
val equal           : nfa * nfa -> bool
val isomorphism     : nfa * nfa * sym_rel -> bool
val findIsomorphism : nfa * nfa -> sym_rel
val isomorphic      : nfa * nfa -> bool
val renameStates    : nfa * sym_rel -> nfa
val renameStatesCanonically : nfa -> nfa
val processStr      : nfa -> sym set * str -> sym set
val accepted       : nfa -> str -> bool
```

26

## (3.9) Processing NFAs in Forlan (Cont.)

More inherited functions:

```
val checkLP      : nfa -> lp -> unit
val validLP      : nfa -> lp -> bool
val findLP       : nfa -> sym set * str * sym set -> lp
val findAcceptingLP : nfa -> str -> lp
val simplify     : nfa -> nfa
```

27

## (3.9) More on Empty-closure Functions

Finally, the functions for computing forwards and backwards empty-closures are inherited by the EFA module

```
val emptyClose      : efa -> sym set -> sym set
val emptyCloseBackwards : efa -> sym set -> sym set
```

and by the NFA module

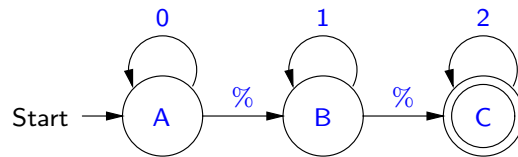
```
val emptyClose      : nfa -> sym set -> sym set
val emptyCloseBackwards : nfa -> sym set -> sym set
```

(of course, the NFA versions of these functions don't do anything interesting).

28

## (3.9) Forlan Examples

Suppose that `efa` is the `efa`



Here are some example uses of a few of the above functions:

```
- projEFAToNFA efa;
invalid label in transition : "%"
```

```
uncaught exception Error
- val nfa = efaToNFA efa;
val nfa = - : nfa
```

29

## (3.9) Forlan Examples (Cont.)

```
- NFA.output("", nfa);
{states}
A, B, C
{start state}
A
{accepting states}
A, B, C
{transitions}
A, 0 -> A | B | C; A, 1 -> B | C; A, 2 -> C;
B, 1 -> B | C; B, 2 -> C; C, 2 -> C
val it = () : unit
```

30

### (3.9) Forlan Examples (Cont.)

```
- LP.output("", EFA.findAcceptingLP efa (Str.input ""));
@ 012
@ .
A, 0 => A, % => B, 1 => B, % => C, 2 => C
val it = () : unit
- LP.output("", NFA.findAcceptingLP nfa (Str.input ""));
@ 012
@ .
A, 0 => A, 1 => B, 2 => C
val it = () : unit
```