

Section 3.6: Simplification of Finite Automata

In this section, we:

- Say what it means for a finite automaton to be simplified;
- Study an algorithm for simplifying finite automata;
- See how finite automata can be simplified in Forlan.

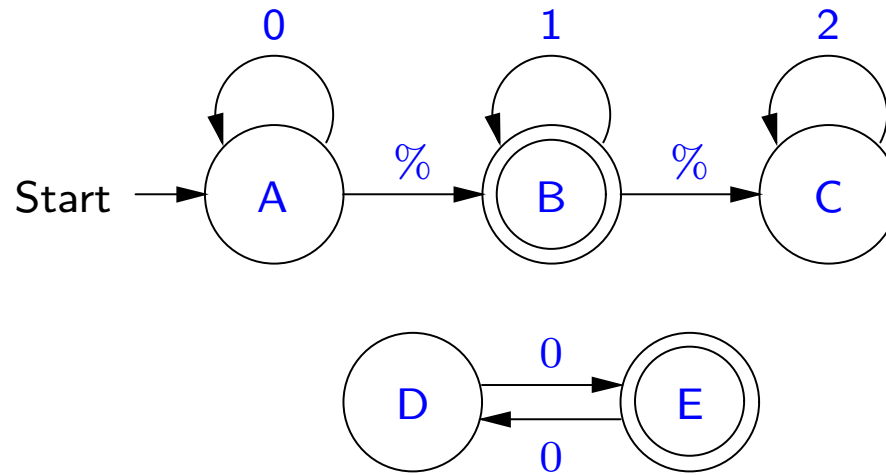
Copyright © 2003–4 Alley Stoughton

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 \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/>.

(3.6) Introduction

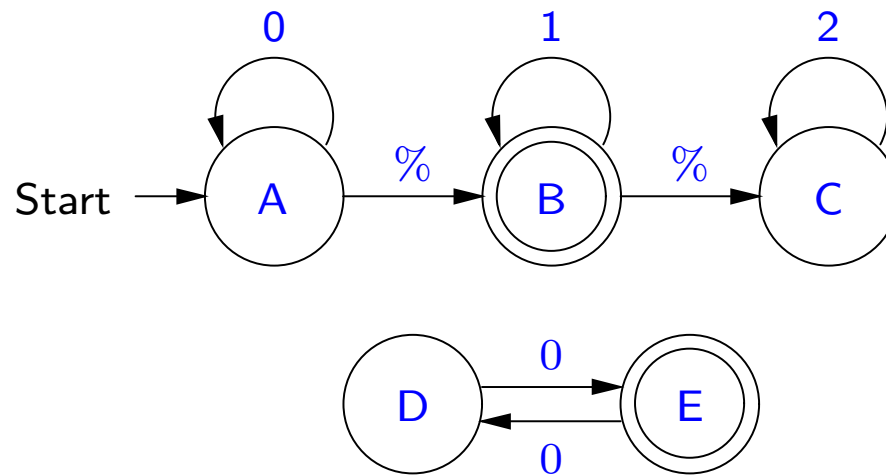
Suppose M is the finite automaton



What is odd about M ?

(3.6) Introduction

Suppose M is the finite automaton



What is odd about M ? There are no valid labeled paths from the start state to D and E , and so these states are redundant; and there are no valid labeled paths from C to an accepting state, and so it is also redundant. We will say that C is not “live” (C is “dead”), and that D and E are not “reachable”.

(3.6) Reachable, Live, Dead and Useful States

Suppose M is a finite automaton. We say that a state $q \in Q_M$ is:

- *reachable* iff there is a labeled path lp such that lp is valid for M , the start state of lp is s_M , and the end state of lp is q ;
- *live* iff there is a labeled path lp such that lp is valid for M , the start state of lp is q , and the end state of lp is in A_M ;
- *dead* iff q is not live;
- *useful* iff q is both reachable and live.

(3.6) Reachable, Live, Dead and Useful States

Suppose M is a finite automaton. We say that a state $q \in Q_M$ is:

- *reachable* iff there is a labeled path lp such that lp is valid for M , the start state of lp is s_M , and the end state of lp is q ;
- *live* iff there is a labeled path lp such that lp is valid for M , the start state of lp is q , and the end state of lp is in A_M ;
- *dead* iff q is not live;
- *useful* iff q is both reachable and live.

Let M be our example finite automaton.

- The reachable states of M are:
- The live states of M are:
- The useful states of M are:

(3.6) Reachable, Live, Dead and Useful States

Suppose M is a finite automaton. We say that a state $q \in Q_M$ is:

- *reachable* iff there is a labeled path lp such that lp is valid for M , the start state of lp is s_M , and the end state of lp is q ;
- *live* iff there is a labeled path lp such that lp is valid for M , the start state of lp is q , and the end state of lp is in A_M ;
- *dead* iff q is not live;
- *useful* iff q is both reachable and live.

Let M be our example finite automaton.

- The reachable states of M are: A , B and C .
- The live states of M are:
- The useful states of M are:

(3.6) Reachable, Live, Dead and Useful States

Suppose M is a finite automaton. We say that a state $q \in Q_M$ is:

- *reachable* iff there is a labeled path lp such that lp is valid for M , the start state of lp is s_M , and the end state of lp is q ;
- *live* iff there is a labeled path lp such that lp is valid for M , the start state of lp is q , and the end state of lp is in A_M ;
- *dead* iff q is not live;
- *useful* iff q is both reachable and live.

Let M be our example finite automaton.

- The reachable states of M are: A , B and C .
- The live states of M are: A , B , D and E .
- The useful states of M are:

(3.6) Reachable, Live, Dead and Useful States

Suppose M is a finite automaton. We say that a state $q \in Q_M$ is:

- *reachable* iff there is a labeled path lp such that lp is valid for M , the start state of lp is s_M , and the end state of lp is q ;
- *live* iff there is a labeled path lp such that lp is valid for M , the start state of lp is q , and the end state of lp is in A_M ;
- *dead* iff q is not live;
- *useful* iff q is both reachable and live.

Let M be our example finite automaton.

- The reachable states of M are: A , B and C .
- The live states of M are: A , B , D and E .
- The useful states of M are: A and B .

(3.6) Checking if a State is Reachable or Live

There is a simple algorithm for generating the set of reachable states of a finite automaton M . We generate the least subset X of Q_M such that:

- $s_M \in X$;
- for all $q, r \in Q_M$ and $x \in \mathbf{Str}$, if $q \in X$ and $(q, x, r) \in T_M$, then $r \in X$.

(3.6) Checking if a State is Reachable or Live

There is a simple algorithm for generating the set of reachable states of a finite automaton M . We generate the least subset X of Q_M such that:

- $s_M \in X$;
- for all $q, r \in Q_M$ and $x \in \mathbf{Str}$, if $q \in X$ and $(q, x, r) \in T_M$, then $r \in X$.

Similarly, there is a simple algorithm for generating the set of live states of a finite automaton M . We generate the least subset Y of Q_M such that:

- $\subseteq Y$;
- for all $q, r \in Q_M$ and $x \in \mathbf{Str}$, if $\in Y$ and $(q, x, r) \in T_M$, then $\in Y$.

(3.6) Checking if a State is Reachable or Live

There is a simple algorithm for generating the set of reachable states of a finite automaton M . We generate the least subset X of Q_M such that:

- $s_M \in X$;
- for all $q, r \in Q_M$ and $x \in \mathbf{Str}$, if $q \in X$ and $(q, x, r) \in T_M$, then $r \in X$.

Similarly, there is a simple algorithm for generating the set of live states of a finite automaton M . We generate the least subset Y of Q_M such that:

- $A_M \subseteq Y$;
- for all $q, r \in Q_M$ and $x \in \mathbf{Str}$, if $r \in Y$ and $(q, x, r) \in T_M$, then $q \in Y$.

(3.6) Checking if a State is Reachable or Live

There is a simple algorithm for generating the set of reachable states of a finite automaton M . We generate the least subset X of Q_M such that:

- $s_M \in X$;
- for all $q, r \in Q_M$ and $x \in \mathbf{Str}$, if $q \in X$ and $(q, x, r) \in T_M$, then $r \in X$.

Similarly, there is a simple algorithm for generating the set of live states of a finite automaton M . We generate the least subset Y of Q_M such that:

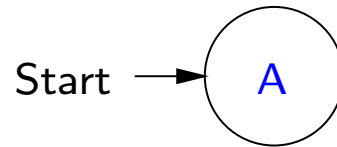
- $A_M \subseteq Y$;
- for all $q, r \in Q_M$ and $x \in \mathbf{Str}$, if $r \in Y$ and $(q, x, r) \in T_M$, then $q \in Y$.

(3.6) Simplified Finite Automata

We say that a finite automaton M is *simplified* iff either

- every state of M is useful; or
- $|Q_M| = 1$ and $|A_M| = |T_M| = 0$.

Let N be the finite automaton



Then N is simplified, even though $s_N = A$ is not live, and thus is not useful.

(3.6) Simplified Finite Automata (Cont.)

Proposition 3.6.1

Suppose M is a simplified finite automaton. Then

$\text{alphabet}(M) = \text{alphabet}(L(M))$.

We always have that $\text{alphabet}(L(M)) \subseteq \text{alphabet}(M)$. But, when M is simplified, we also have that

$\text{alphabet}(M) \subseteq \text{alphabet}(L(M))$, i.e., that every symbol appearing in a string of one of M 's transitions also appears in one of the strings accepted by M .

(3.6) Simplified Finite Automata (Cont.)

We define a function $\mathbf{simplify} \in \mathbf{FA} \rightarrow \mathbf{FA}$ by: $\mathbf{simplify}(M)$ is the finite automaton N such that:

- if s_M is useful in M , then:
 - $Q_N = \{q \in Q_M \mid q \text{ is useful in } M\}$;
 - $s_N = s_M$;
 - $A_N = A_M \cap Q_N = \{q \in A_M \mid q \in Q_N\}$;
 - $T_N = \{(q, x, r) \in T_M \mid q, r \in Q_N\}$; and
- if s_M is not useful in M , then:
 - $Q_N = \{s_M\}$;
 - $s_N = s_M$;
 - $A_N = \emptyset$;
 - $T_N = \emptyset$.

(3.6) Simplified Finite Automata (Cont.)

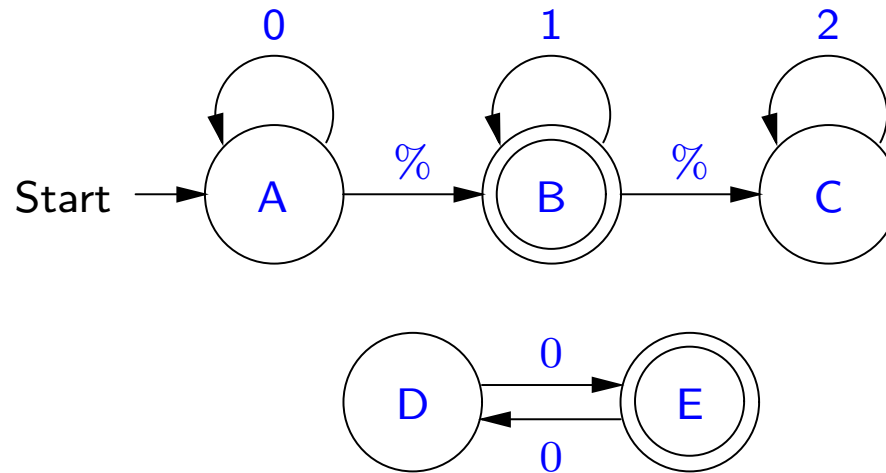
Proposition 3.6.2

Suppose M is a finite automaton. Then:

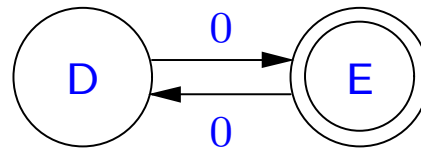
- (1) $\text{simplify}(M)$ is simplified;
- (2) $\text{simplify}(M) \approx M$;
- (3) $\text{alphabet}(\text{simplify}(M)) \subseteq \text{alphabet}(M)$.

(3.6) Simplified Finite Automata (Cont.)

Suppose M is the finite automaton

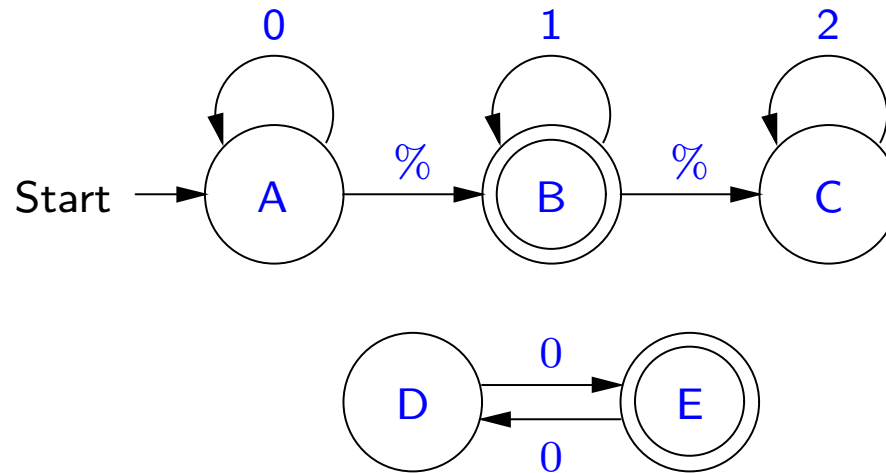


Then $\text{simplify}(M)$ is the finite automaton

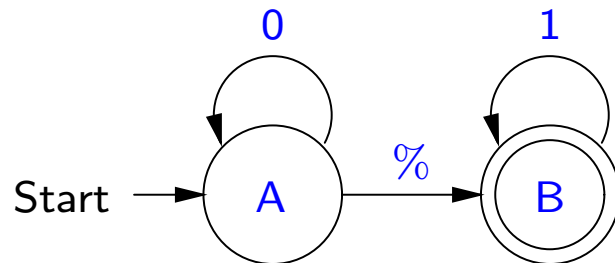


(3.6) Simplified Finite Automata (Cont.)

Suppose M is the finite automaton



Then $\text{simplify}(M)$ is the finite automaton

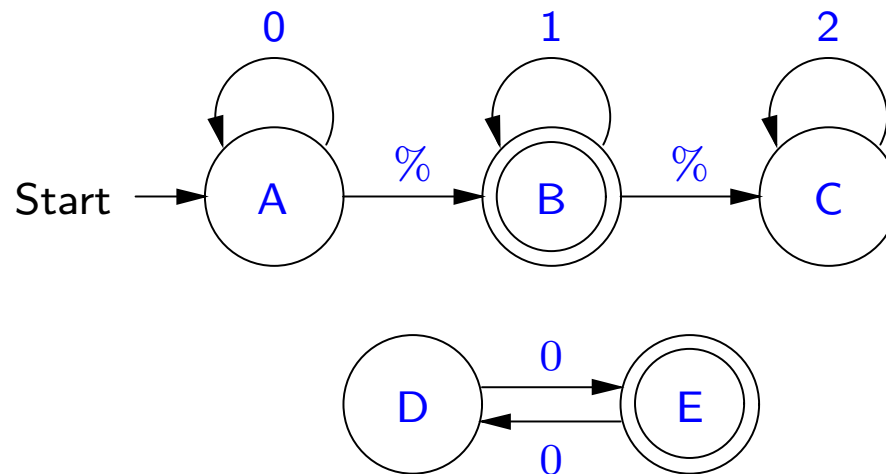


(3.6) Simplification of Finite Automata in Forlan

The Forlan module `FA` includes the following function for simplifying finite automata:

```
val simplify : fa -> fa
```

In the following, suppose `fa` is the finite automaton



(3.6) Forlan Examples

Here are some example uses of `simplify`:

```
- val fa' = FA.simplify fa;
val fa' = - : fa
- FA.output("", fa');
{states}
A, B
{start state}
A
{accepting states}
B
{transitions}
A, % -> B; A, 0 -> A; B, 1 -> B
val it = () : unit
```

(3.6) Forlan Examples (Cont.)

More examples:

```
- val fa'' = FA.input "";
@ {states} A, B {start state} A {accepting states}
@ {transitions} A, 0 -> B; B, 0 -> A
@ .
val fa'' = - : fa
- FA.output("", FA.simplify fa'');
{states}
A
{start state}
A
{accepting states}

{transitions}

val it = () : unit
```