

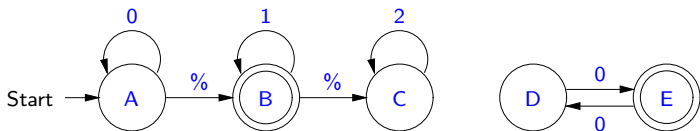
### *3.7: Simplification of Finite Automata*

In this section, we: say what it means for a finite automaton to be simplified; study an algorithm for simplifying finite automata; and see how finite automata can be simplified in Forlan.

### 3.7: 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; and see how finite automata can be simplified in Forlan.

Suppose  $M$  is the finite automaton

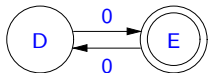
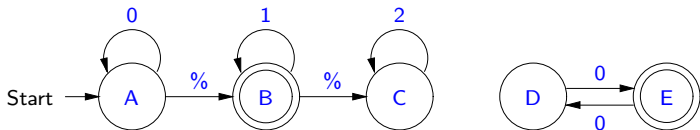


What is odd about  $M$ ?

### 3.7: 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; and see how finite automata can be simplified in Forlan.

Suppose  $M$  is the finite automaton



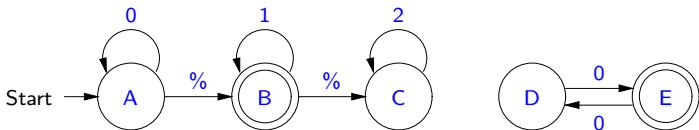
What is odd about  $M$ ?

First, there are no valid labeled paths from the start state to D and E, and so these states are redundant.

### 3.7: 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; and see how finite automata can be simplified in Forlan.

Suppose  $M$  is the finite automaton



What is odd about  $M$ ?

First, there are no valid labeled paths from the start state to  $D$  and  $E$ , and so these states are redundant.

Second, there are no valid labeled paths from  $C$  to an accepting state, and so it is also redundant.

## Useful States

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

- *reachable in  $M$*  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  $q$ ;

## Useful States

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

- *reachable in  $M$*  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 in  $M$*  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  $A_M$ ;

## Useful States

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

- *reachable in  $M$*  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 in  $M$*  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 in  $M$*  iff  $q$  is not live in  $M$ ; and

## Useful States

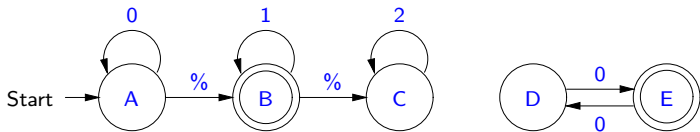
Suppose  $M$  is a finite automaton. We say that a state  $q \in Q_M$  is:

- *reachable in  $M$*  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 in  $M$*  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 in  $M$*  iff  $q$  is not live in  $M$ ; and
- *useful in  $M$*  iff  $q$  is both reachable and live in  $M$ .



## Useful States Example

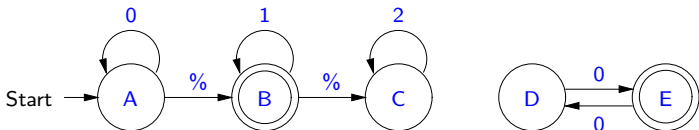
Let  $M$  be our example finite automaton:



The reachable states of  $M$  are:

## Useful States Example

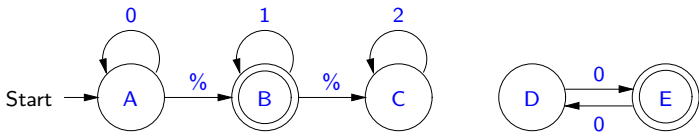
Let  $M$  be our example finite automaton:



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

## Useful States Example

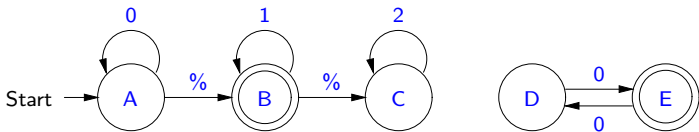
Let  $M$  be our example finite automaton:



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

## Useful States Example

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  $E$ . And, the useful states of  $M$  are:  $A$  and  $B$ .

## *Generating Reachable, Live and Useful States*

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:

- $q_0 \in X$ ; and
- for all  $q, r \in Q_M$  and  $x \in \mathbf{Str}$ , if  $q \in X$  and  $(q, x, r) \in T_M$ , then  $r \in X$ .

## *Generating Reachable, Live and Useful States*

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:

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

## *Generating Reachable, Live and Useful States*

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:

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

## *Generating Reachable, Live and Useful States*

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:

- $s_M \in X$ ; and
- 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:

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



## *Generating Reachable, Live and Useful States*

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:

- $s_M \in X$ ; and
- 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:

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

## *Generating Reachable, Live and Useful States*

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:

- $s_M \in X$ ; and
- 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:

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

## *Generating Reachable, Live and Useful States*

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:

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

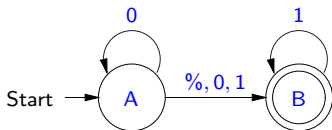
Similarly, there is a simple algorithm for generating the set of live states of a finite automaton  $M$ . We generate the least subset  $Y$  of  $Q_M$  such that:

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

Thus, we can generate the set of useful states of an FA by generating the set of reachable states, generating the set of live states, and intersecting those sets of states.

## *Redundant Transitions*

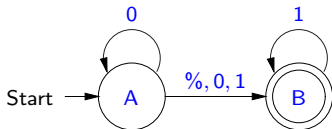
Now, suppose  $N$  is the FA



What is odd about this machine?

## Redundant Transitions

Now, suppose  $N$  is the FA

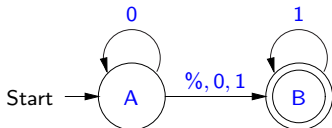


What is odd about this machine?

Here, the transitions  $(A, 0, B)$  and  $(A, 1, B)$  are redundant.

## Redundant Transitions

Now, suppose  $N$  is the FA



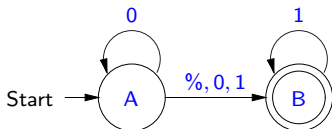
What is odd about this machine?

Here, the transitions  $(A, 0, B)$  and  $(A, 1, B)$  are redundant.

Given an FA  $M$  and a finite subset  $U$  of  $\{(q, x, r) \mid q, r \in Q_M \text{ and } x \in \text{Str}\}$ , we write  $M/U$  for the FA that is identical to  $M$  except that its set of transitions is  $U$ .

## Redundant Transitions

Now, suppose  $N$  is the FA



What is odd about this machine?

Here, the transitions  $(A, 0, B)$  and  $(A, 1, B)$  are redundant.

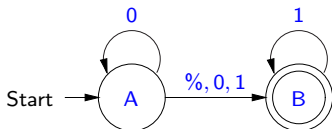
Given an FA  $M$  and a finite subset  $U$  of  $\{(q, x, r) \mid q, r \in Q_M \text{ and } x \in \text{Str}\}$ , we write  $M/U$  for the FA that is identical to  $M$  except that its set of transitions is  $U$ .

If  $M$  is an FA and  $(p, x, q) \in T_M$ , we say that:

- $(p, x, q)$  is redundant in  $M$  iff  $q \in \Delta_N(\{p\}, x)$ , where  $N = M/(T_M - \{(p, x, q)\})$ ; and

## Redundant Transitions

Now, suppose  $N$  is the FA



What is odd about this machine?

Here, the transitions  $(A, 0, B)$  and  $(A, 1, B)$  are redundant.

Given an FA  $M$  and a finite subset  $U$  of  $\{(q, x, r) \mid q, r \in Q_M \text{ and } x \in \mathbf{Str}\}$ , we write  $M/U$  for the FA that is identical to  $M$  except that its set of transitions is  $U$ .

If  $M$  is an FA and  $(p, x, q) \in T_M$ , we say that:

- $(p, x, q)$  is *redundant* in  $M$  iff  $q \in \Delta_N(\{p\}, x)$ , where  $N = M/(T_M - \{(p, x, q)\})$ ; and
- $(p, x, q)$  is *irredundant* in  $M$  iff  $(p, x, q)$  is not redundant in  $M$ .



## *Definition of Simplification*

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

- every state of  $M$  is useful, and every transition of  $M$  is irredundant; or

## *Definition of Simplification*

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

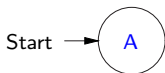
- every state of  $M$  is useful, and every transition of  $M$  is irredundant; or
- $|Q_M| = 1$  and  $A_M = T_M = \emptyset$ .

## Definition of Simplification

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

- every state of  $M$  is useful, and every transition of  $M$  is irredundant; or
- $|Q_M| = 1$  and  $A_M = T_M = \emptyset$ .

Thus the FA

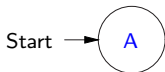


## Definition of Simplification

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

- every state of  $M$  is useful, and every transition of  $M$  is irredundant; or
- $|Q_M| = 1$  and  $A_M = T_M = \emptyset$ .

Thus the FA



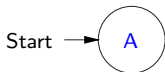
is simplified, even though its start state is not live, and is thus not useful.

## Definition of Simplification

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

- every state of  $M$  is useful, and every transition of  $M$  is irredundant; or
- $|Q_M| = 1$  and  $A_M = T_M = \emptyset$ .

Thus the FA



is simplified, even though its start state is not live, and is thus not useful.

### Proposition 3.7.1

If  $M$  is a simplified finite automaton, then  
**alphabet**  $M = \mathbf{alphabet}(L(M))$ .

## *Algorithm for Removing Redundant Transitions*

Given an FA  $M$ ,  $p, q \in Q_M$  and  $x \in \mathbf{Str}$ , we say that  $(p, x, q)$  is *implicit in  $M$*  iff  $q \in \Delta_M(\{p\}, x)$ .

## Algorithm for Removing Redundant Transitions

Given an FA  $M$ ,  $p, q \in Q_M$  and  $x \in \mathbf{Str}$ , we say that  $(p, x, q)$  is *implicit in  $M$*  iff  $q \in \Delta_M(\{p\}, x)$ .

Given an FA  $M$ , we define a function

$\mathbf{remRedun}_M \in \mathcal{P} T_M \times \mathcal{P} T_M \rightarrow \mathcal{P} T_M$  by well-founded recursion on the size of its second argument.

For  $U, V \subseteq T_M$ ,  $\mathbf{remRedun}(U, V)$  proceeds as follows:

## Algorithm for Removing Redundant Transitions

Given an FA  $M$ ,  $p, q \in Q_M$  and  $x \in \mathbf{Str}$ , we say that  $(p, x, q)$  is *implicit in  $M$*  iff  $q \in \Delta_M(\{p\}, x)$ .

Given an FA  $M$ , we define a function

$\mathbf{remRedun}_M \in \mathcal{P} T_M \times \mathcal{P} T_M \rightarrow \mathcal{P} T_M$  by well-founded recursion on the size of its second argument.

For  $U, V \subseteq T_M$ ,  $\mathbf{remRedun}(U, V)$  proceeds as follows:

- If  $V = \emptyset$ , then it returns  $U$ .



## Algorithm for Removing Redundant Transitions

Given an FA  $M$ ,  $p, q \in Q_M$  and  $x \in \mathbf{Str}$ , we say that  $(p, x, q)$  is *implicit in  $M$*  iff  $q \in \Delta_M(\{p\}, x)$ .

Given an FA  $M$ , we define a function

$\mathbf{remRedun}_M \in \mathcal{P} T_M \times \mathcal{P} T_M \rightarrow \mathcal{P} T_M$  by well-founded recursion on the size of its second argument.

For  $U, V \subseteq T_M$ ,  $\mathbf{remRedun}(U, V)$  proceeds as follows:

- If  $V = \emptyset$ , then it returns  $U$ .
- Otherwise, let  $v$  be the greatest element of  $V$ , and  $V' = V - \{v\}$ .

## Algorithm for Removing Redundant Transitions

Given an FA  $M$ ,  $p, q \in Q_M$  and  $x \in \mathbf{Str}$ , we say that  $(p, x, q)$  is *implicit in  $M$*  iff  $q \in \Delta_M(\{p\}, x)$ .

Given an FA  $M$ , we define a function

$\mathbf{remRedun}_M \in \mathcal{P} T_M \times \mathcal{P} T_M \rightarrow \mathcal{P} T_M$  by well-founded recursion on the size of its second argument.

For  $U, V \subseteq T_M$ ,  $\mathbf{remRedun}(U, V)$  proceeds as follows:

- If  $V = \emptyset$ , then it returns  $U$ .
- Otherwise, let  $v$  be the greatest element of  $V$ , and  $V' = V - \{v\}$ . If  $v$  is implicit in  $M/(U \cup V')$ , then  $\mathbf{remRedun}$  returns the result of evaluating  $\mathbf{remRedun}(U, V')$ .

## Algorithm for Removing Redundant Transitions

Given an FA  $M$ ,  $p, q \in Q_M$  and  $x \in \mathbf{Str}$ , we say that  $(p, x, q)$  is *implicit in  $M$*  iff  $q \in \Delta_M(\{p\}, x)$ .

Given an FA  $M$ , we define a function

$\mathbf{remRedun}_M \in \mathcal{P} T_M \times \mathcal{P} T_M \rightarrow \mathcal{P} T_M$  by well-founded recursion on the size of its second argument.

For  $U, V \subseteq T_M$ ,  $\mathbf{remRedun}(U, V)$  proceeds as follows:

- If  $V = \emptyset$ , then it returns  $U$ .
- Otherwise, let  $v$  be the greatest element of  $V$ , and  $V' = V - \{v\}$ . If  $v$  is implicit in  $M/(U \cup V')$ , then  $\mathbf{remRedun}$  returns the result of evaluating  $\mathbf{remRedun}(U, V')$ . Otherwise, it returns the result of evaluating  $\mathbf{remRedun}(U \cup \{v\}, V')$ .

## Algorithm for Removing Redundant Transitions

Given an FA  $M$ ,  $p, q \in Q_M$  and  $x \in \mathbf{Str}$ , we say that  $(p, x, q)$  is *implicit in  $M$*  iff  $q \in \Delta_M(\{p\}, x)$ .

Given an FA  $M$ , we define a function

**remRedun** $_M \in \mathcal{P} T_M \times \mathcal{P} T_M \rightarrow \mathcal{P} T_M$  by well-founded recursion on the size of its second argument.

For  $U, V \subseteq T_M$ , **remRedun** $(U, V)$  proceeds as follows:

- If  $V = \emptyset$ , then it returns  $U$ .
- Otherwise, let  $v$  be the greatest element of  $V$ , and  $V' = V - \{v\}$ . If  $v$  is implicit in  $M/(U \cup V')$ , then **remRedun** returns the result of evaluating **remRedun** $(U, V')$ . Otherwise, it returns the result of evaluating **remRedun** $(U \cup \{v\}, V')$ .

In general, there are multiple—incompatible—ways of removing redundant transitions from an FA. **remRedun** is defined so as to favor removing transitions that are larger in our total ordering on transitions.

## *Simplification Algorithm*

We define a function **simplify**  $\in \mathbf{FA} \rightarrow \mathbf{FA}$  by: **simplify**  $M$  is the finite automaton  $N$  produced by the following process.

- First, the useful states are  $M$  are determined.

## *Simplification Algorithm*

We define a function **simplify**  $\in \mathbf{FA} \rightarrow \mathbf{FA}$  by: **simplify**  $M$  is the finite automaton  $N$  produced by the following process.

- First, the useful states are  $M$  are determined.
- If  $s_M$  is not useful in  $M$ , the  $N$  is defined by:
  - $Q_N = \{s_M\}$ ;
  - $s_N = s_M$ ;
  - $A_N = \emptyset$ ; and
  - $T_N = \emptyset$ .

## *Simplification Algorithm*

We define a function **simplify**  $\in \mathbf{FA} \rightarrow \mathbf{FA}$  by: **simplify**  $M$  is the finite automaton  $N$  produced by the following process.

- First, the useful states are  $M$  are determined.
- If  $s_M$  is not useful in  $M$ , the  $N$  is defined by:
  - $Q_N = \{s_M\}$ ;
  - $s_N = s_M$ ;
  - $A_N = \emptyset$ ; and
  - $T_N = \emptyset$ .
- And, if  $s_M$  is useful in  $M$ , then  $N$  is defined by:
  - $Q_N = \{q \in Q_M \mid q \text{ is useful in } M\}$ ;
  - $s_N = s_M$ ;
  - $A_N = A_M \cap Q_N = \{q \in A_M \mid q \in Q_N\}$ ; and
  - $T_N = \mathbf{remRedun}(\emptyset, \{(q, x, r) \in T_M \mid q, r \in Q_N\})$ .

## More on Simplification Algorithm

### Proposition 3.7.3

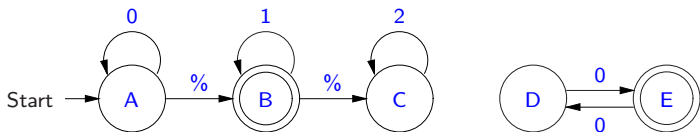
Suppose  $M$  is a finite automaton. Then:

- (1) **simplify**  $M$  is simplified;
- (2) **simplify**  $M \approx M$ ; and
- (3) **alphabet**(**simplify**  $M$ ) = **alphabet**( $L(M)$ )  $\subseteq$  **alphabet**  $M$ .



## *Simplification Examples*

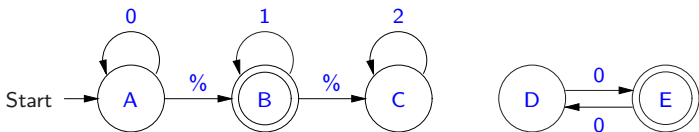
If  $M$  is the finite automaton



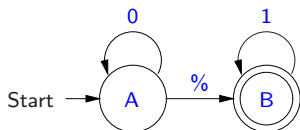
then **simplify**  $M$  is the finite automaton

## *Simplification Examples*

If  $M$  is the finite automaton

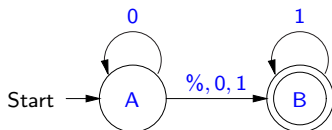


then **simplify**  $M$  is the finite automaton



## *Simplification Examples*

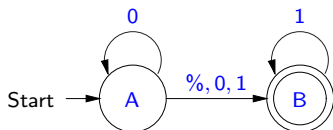
If  $N$  is the finite automaton



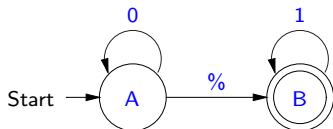
then **simplify**  $N$  is the finite automaton

## *Simplification Examples*

If  $N$  is the finite automaton



then **simplify**  $N$  is the finite automaton



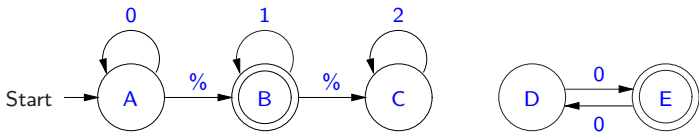
## *Simplification in Forlan*

The Forlan module FA includes the following functions relating to the simplification of finite automata:

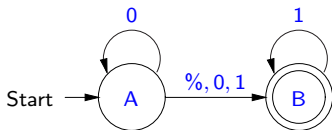
```
val simplify    : fa -> fa  
val simplified  : fa -> bool
```

## *Simplification Examples in Forlan*

In the following, suppose **fa1** is the finite automaton

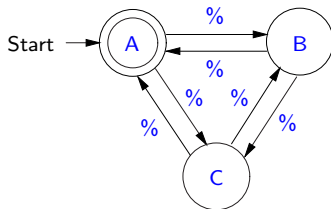


**fa2** is the finite automaton



## *Simplification Examples in Forlan*

and `fa3` is the finita automaton



## *Simplification Examples in Forlan*

Here are some example uses of `simplify` and `simplified`:

```
- FA.simplified fa1;
val it = false : bool
- val fa1' = FA.simplify fa1;
val fa1' = - : fa
- FA.output("", fa1');
{states} A, B {start state} A {accepting states} B
{transitions} A, % -> B; A, 0 -> A; B, 1 -> B
val it = () : unit
- FA.simplified fa1';
val it = true : bool
- val fa2' = FA.simplify fa2;
val fa2' = - : fa
- FA.output("", fa2');
{states} A, B {start state} A {accepting states} B
{transitions} A, % -> B; A, 0 -> A; B, 1 -> B
val it = () : unit
```



## *Simplification Examples in Forlan*

```
- val fa3' = FA.simplify fa3;  
val fa3' = - : fa  
- FA.output("", fa3');  
{states} A, B, C {start state} A {accepting states} A  
{transitions} A, % -> B | C; B, % -> A; C, % -> A  
val it = () : unit
```

Thus the simplification of `fa3` resulted in the removal of the %-transitions between `B` and `C`.