3.4: Finite Automata and Labeled Paths

In this section, we:

- say what finite automata (FA) are, and give an introduction to how they can be processed using Forlan;
- say what labeled paths are, and show how they can be processed using Forlan; and
- use the notion of labeled path to say what finite automata mean.

Finite Automata

A *finite automaton* (FA) M consists of:

- a finite set Q_M of symbols (we call the elements of Q_M the *states* of M);
- an element s_M of Q_M (we call s_M the *start state* of M);
- a subset A_M of Q_M (we call the elements of A_M the *accepting states* of M);
- a finite subset T_M of $\{ (q, x, r) \mid q, r \in Q_M \text{ and } x \in \mathbf{Str} \}$ (we call the elements of T_M the *transitions* of M, and we often write (q, x, r) as

$$q \xrightarrow{x} r$$

or $q, x \to r$).

Finite Automata

We often abbreviate Q_M, s_M, A_M and T_M to Q, s, A and T, when it's clear which FA we are working with.

We write **FA** for the set of all finite automata, which is a countably infinite set.

Example FA

As an example, we can define an FA M as follows:

- $Q_M = \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$;
- $s_M = \mathsf{A}$;
- $A_M = \{\mathsf{A}, \mathsf{C}\}$;
- $T_M = \{(\mathsf{A}, 1, \mathsf{A}), (\mathsf{B}, 11, \mathsf{B}), (\mathsf{C}, 111, \mathsf{C}), (\mathsf{A}, 0, \mathsf{B}), (\mathsf{A}, 2, \mathsf{B}), (\mathsf{A}, 0, \mathsf{C}), (\mathsf{A}, 2, \mathsf{C}), (\mathsf{B}, 0, \mathsf{C}), (\mathsf{B}, 2, \mathsf{C})\}$.

Forlan's Syntax for FAs

Here is how our example FA M can be expressed in Forlan's syntax:

```
{states}
A, B, C
{start state}
A
{accepting states}
A, C
{transitions}
A, 1 -> A; B, 11 -> B; C, 111 -> C;
A, 0 -> B; A, 2 -> B;
A, 0 -> C; A, 2 -> C;
B, 0 -> C; B, 2 -> C
```

Forlan's Syntax for FAs

Since whitespace characters are ignored by Forlan's input routines, the preceding description of *M* could have been formatted in many other ways. States are separated by commas, and transitions are separated by semicolons. The order of states and transitions is irrelevant.

Transitions that only differ in their right-hand states can be merged into single transition families. E.g., we can merge

```
A, 0 -> B
```

and

```
A, 0 -> C
```

into the transition family

```
A, 0 -> B | C
```

Input and Output of FAs in Forlan

The Forlan module `FA` defines an abstract type `fa` (in the top-level environment) of finite automata, as well as a large number of functions and constants for processing FAs, including:

```
val input  : string -> fa
val output : string * fa -> unit
```

I/O Example

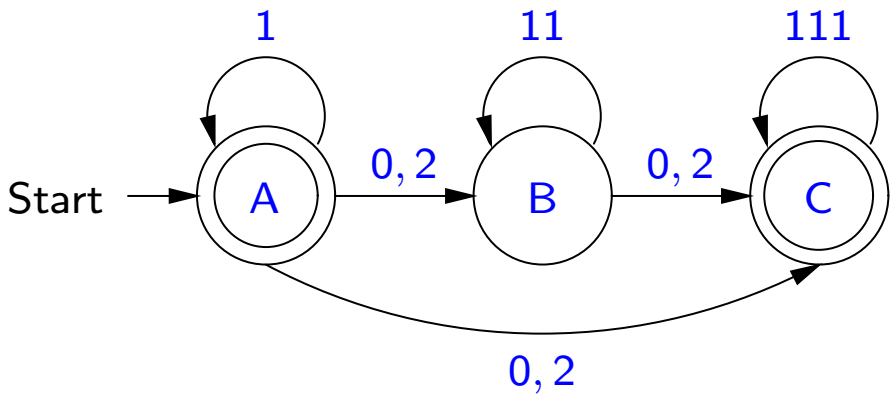
Suppose that our example FA is in the file `3.4-fa`. We can input this FA into Forlan, and then output it to the standard output, as follows:

```
- val fa = FA.input "3.4-fa";
val fa = - : fa
- FA.output("", fa);
{states} A, B, C {start state} A
{accepting states} A, C
{transitions}
A, 0 -> B | C; A, 1 -> A; A, 2 -> B | C; B, 0 -> C;
B, 2 -> C; B, 11 -> B; C, 111 -> C
val it = () : unit
```

Graphical Notation for FAs

We also make use of graphical notation for finite automata.

Here is how our FA M can be described graphically:



Graphical Editor for Finite Automata

The Java program JForlan, can be used to view and edit finite automata. It can be invoked directly, or run via Forlan. See the Forlan website for more information.

More on Finite Automata

We define a function **alphabet** $\in$ **FA** $\to$ **Alp** by: for all $M \in$ **FA**, **alphabet** M is $\{\, a \in$ **Sym** $\mid$ there are q, x, r such that $q, x \to r \in T_M$ and $a \in$ **alphabet** $x \,\}$.

For example, the alphabet of our example FA M is $\{0, 1, 2\}$.

The Forlan module `FA` contains the functions

```
val numStates      : fa -> int
val numTransitions : fa -> int
val equal          : fa * fa -> bool
val alphabet       : fa -> sym set
```

Labeled Paths and FA Meaning

We will explain when strings are accepted by finite automata using the notion of a labeled path. A *labeled path* lp has the form

$$q_1 \overset{x_1}{\Rightarrow} q_2 \overset{x_2}{\Rightarrow} \cdots q_n \overset{x_n}{\Rightarrow} q_{n+1},$$

where the q_i's (which we think of as states) are symbols, and the x_i's are strings. (When $n = 0$, this is just $q_{0+1} = q_1$.)

We write **LP** for the set of all labeled paths, which is a countably infinite set.

We sometimes (e.g., when using Forlan) write a path

$$q_1 \overset{x_1}{\Rightarrow} q_2 \overset{x_2}{\Rightarrow} \cdots q_n \overset{x_n}{\Rightarrow} q_{n+1}$$

as

$$q_1, x_1 \Rightarrow q_2, x_2 \Rightarrow \cdots q_n, x_n \Rightarrow q_{n+1}.$$

Notation for Labeled Paths

Let lp be the labeled path

$$q_1 \overset{x_1}{\Rightarrow} q_2 \overset{x_2}{\Rightarrow} \cdots q_n \overset{x_n}{\Rightarrow} q_{n+1},$$

We say that:

- the *start state* of lp (**startState** lp) is q_1;
- the *end state* of lp (**endState** lp) is q_{n+1};
- the *length* of lp ($|lp|$) is n; and
- the *label* of lp (**label** lp) is $x_1 x_2 \cdots x_n$ ($\%$, when $n = 0$).

Example Labeled Paths

For example

$$\mathsf{A}$$

is a labeled path whose start and end states are both A, whose length is 0, and whose label is $\%$. And

$$\mathsf{A} \overset{0}{\Rightarrow} \mathsf{B} \overset{11}{\Rightarrow} \mathsf{B} \overset{2}{\Rightarrow} \mathsf{C}$$

is a labeled path whose

- start state is A;
- end state is C;
- length is 3; and
- label is $0(11)2 = 0112$.

Validity of Labeled Paths in FAs

A labeled path

$$q_1 \overset{x_1}{\Rightarrow} q_2 \overset{x_2}{\Rightarrow} \cdots q_n \overset{x_n}{\Rightarrow} q_{n+1},$$

is *valid for* an FA M iff, for all $i \in [1 : n]$,

$$q_i, x_i \to q_{i+1} \in T_M,$$

and $q_{n+1} \in Q_M$.

For example, the labeled paths

$$\mathsf{A} \qquad \text{and} \qquad \mathsf{A} \overset{0}{\Rightarrow} \mathsf{B} \overset{11}{\Rightarrow} \mathsf{B} \overset{2}{\Rightarrow} \mathsf{C}$$

are valid for our example FA M. But the labeled path

$$\mathsf{A} \overset{\%}{\Rightarrow} \mathsf{A}$$

is not valid for M, since $\mathsf{A}, \% \to \mathsf{A} \notin T_M$.

The Meaning of Finite Automata

A string w is *accepted by* a finite automaton M iff there is a labeled path lp such that

- lp is valid for M;
- the label of lp is w;
- the start state of lp is the start state of M; and
- the end state of lp is an accepting state of M.

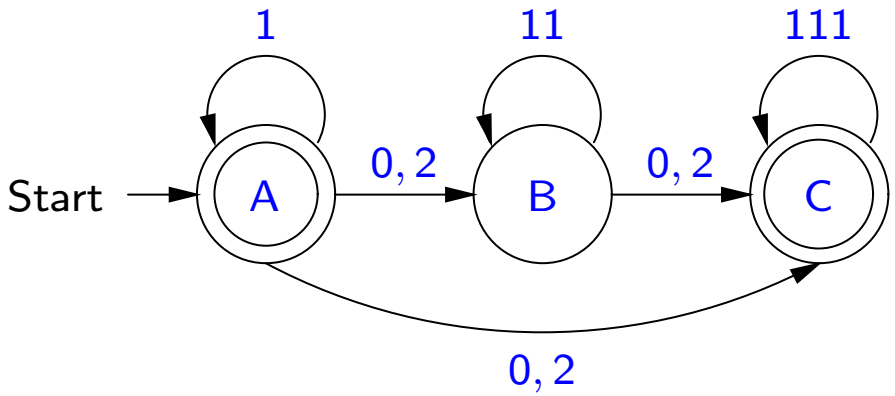
Clearly, if w is accepted by M, then $\textbf{alphabet}\, w \subseteq \textbf{alphabet}\, M$.

The *language accepted by* a finite automaton M ($L(M)$) is

$$\{\, w \in \textbf{Str} \mid w \text{ is accepted by } M \,\}.$$

Example of FA Meaning

Consider our example FA M:



We have that

$$L(M) = \{1\}^* \cup \{1\}^*\{0,2\}\{11\}^*\{0,2\}\{111\}^* \cup \{1\}^*\{0,2\}\{111\}^*.$$

For example, %, 11, 110112111 and 2111111 are accepted by M. And 21112 and 2211 are not accepted by M.

More on Finite Automata Meaning

Proposition 3.4.1
Suppose M *is a finite automaton. Then* **alphabet**$(L(M)) \subseteq$ **alphabet** M.

We say that finite automata M and N are *equivalent* iff $L(M) = L(N)$. In other words, M and N are equivalent iff M and N accept the same language.

We define a relation $\approx$ on **FA** by: $M \approx N$ iff M and N are equivalent. It is easy to see that $\approx$ is reflexive on **FA**, symmetric and transitive.

Processing Labeled Paths in Forlan

The Forlan module `LP` defines an abstract type `lp` (in the top-level environment) of labeled paths, as well as various functions for processing labeled paths, including:

```
val input      : string -> lp
val output     : string * lp -> unit
val equal      : lp * lp -> bool
val startState : lp -> sym
val endState   : lp -> sym
val label      : lp -> str
val length     : lp -> int
```

Checking the Validity of Labeled Paths

The module `FA` also defines the functions

```
val checkLP : fa -> lp -> unit
val validLP : fa -> lp -> bool
```

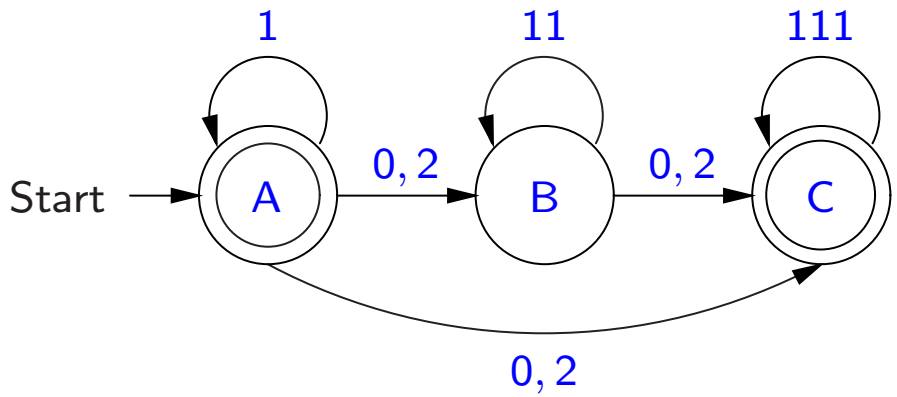
for checking whether a labeled path is valid in a finite automaton.

The function `checkLP` takes in an FA *M* and returns a function that checks whether a labeled path *lp* is valid for *M*. When *lp* is not valid for *M*, the function explains why it isn't; otherwise, it prints nothing.

The function `validLP` takes in an FA *M* and returns a function that tests whether a labeled path *lp* is valid for *M*, silently returning `true`, if it is, and silently returning `false`, otherwise.

Example Labeled Path and FA Processing

Assume that `fa` is still bound to our example FA



Here are some examples of labeled path and FA processing:

Example Labeled Path and FA Processing

```
- val lp = LP.input "";
@ A, 1 => A, 0 => B, 11 => B, 2 => C, 111 => C
@ .
val lp = - : lp
- Sym.output("", LP.startState lp);
A
val it = () : unit
- Sym.output("", LP.endState lp);
C
val it = () : unit
- LP.length lp;
val it = 5 : int
```

Examples

```
- Str.output("", LP.label lp);
10112111
val it = () : unit
- val checkLP = FA.checkLP fa;
val checkLP = fn : lp -> unit
- checkLP lp;
val it = () : unit
- val lp' = LP.fromString "A";
val lp' = - : lp
- LP.length lp';
val it = 0 : int
- Str.output("", LP.label lp');
%
val it = () : unit
- checkLP lp';
val it = () : unit
```

Examples

```
- checkLP(LP.input "");
@ A, % => A, 1 => A
@ .
invalid transition: "A, % -> A"

uncaught exception Error
```

Designing Finite Automata

Let's consider the problem of finding a finite automaton that accepts the set of all strings of 0's and 1's with an even number of 0's.

It seems reasonable that our machine have two states: an accepting (and start) state A corresponding to the strings of 0's and 1's with an even number of zeros, and a state B corresponding to the strings of 0's and 1's with an odd number of zeros. Processing a 1 in either state should cause us to stay in that state, but processing a 0 in one of the states should cause us to switch to the other state.

The above considerations lead us to the FA: